Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

NoSQL Cassandra: Um Estudo de Caso Com Workflows de Bioinformática

Matheus Moreira Marques de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientadora
Prof.ª Dr.ª Maristela Terto de Holanda

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Engenharia da Computação

Coordenador: Prof. Dr. Ricardo Jacobi

Banca examinadora composta por:

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora) — CIC/UnB
Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo — CIC/UnB
Prof. Dr. Marcio de Carvalho Victorino — CIC/UnB

CIP — Catalogação Internacional na Publicação

Oliveira, Matheus Moreira Marques de.

NoSQL Cassandra: Um Estudo de Caso Com Workflows de Bioinformática / Matheus Moreira Marques de Oliveira. Brasília : UnB, 2016. 68 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. Bioinformática, 2. Cassandra, 3. NoSQL, 4. Workflow.

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

NoSQL Cassandra: Um Estudo de Caso Com Workflows de Bioinformática

Matheus Moreira Marques de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo
CIC/UnB

Prof. Dr. Marcio de Carvalho Victorino
CIC/UnB

Prof. Dr. Ricardo Jacobi
Coordenador do Curso de Engenharia da Computação

Brasília, 01 de junho de 2016

Dedicatória

Dedico este trabalho aos meus avós paternos e maternos, Tânia, Milton, Delaíde e Francisca, in memorian, e à meus pais, Iolanda e Marcus, pois sem eles este trabalho e muitos dos meus sonhos não se realizariam.

Agradecimentos

É difícil agradecer todas as pessoas que de algum modo, nos momentos tranquilos e ou turbulentos, fizeram ou fazem parte da minha vida, por isso primeiramente agradeço à todos de coração.

Gostaria de agradecer aos meus pais por todo o amor dado durante toda a minha jornada nesta fase da minha vida. Desde o incentivo a buscar o curso de meu interesse, até todo suporte necessário e puxões de orelha.

Obrigado mãe por sempre me colocar na reta e por cuidar tão bem de mim, seus cuidados e dedicação foram essenciais e, em diversos momentos, foram a esperança para seguir em frente.

Obrigado pai por todo o carinho e apoio, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

Gostaria de agradecer à minha irmã pelos nossos dias juntos, que apesar de sempre turbulentos, são divertidos. Obrigado pelas músicas que você me passa e por me ajudar em coisas que não confio mais em ninguém para fazer (tipo fazer minha sobrancelha).

Agradeço à minha vó Francisca, in memorian, por ser um elo do meu caráter e pureza. Muito obrigado pelas memória e ensinamentos. Muito obrigado por me fazer sempre lembrar que, mesmo com as coisas mais simples, podemos fazer coisas maravilhosas (sempre vou amar seu presépio). Você ainda é meu contato favorito no Skype...

Agradeço à minha vó Delaíde e meu vó Milton pelo meu primeiro notebook, material essencial durante minha jornada em um curso de computação, e pelos adoráveis dias que tenho quando vou visitá-los.

Agradeço à minha vó Tânia por todo carinho, pelo suporte sempre oferecido e por sempre estar presente em todos os momentos da minha vida desde que eu me lembro por gente. Gostaria de agradecer também por todos as refeições deliciosas que tive em sua casa e aos cochilos que podia tirar logo após o almoço, eles eram essenciais para não dormir nas aulas das 14h.

Agradeço também a todos meus amigos que me acompanharam nesse curso. Todos aqueles que tiveram a paciência em me ajudar nas matérias mais difíceis e à todos que estiveram comigo até várias horas da noite, seja estudando para um prova ou fazendo

algum trabalho. Gostaria muito de agradecer por cada conversa que tive com vários de vocês, seja tacando pedra no lago, ou durante uma volta no grande-circular, ou durante uma virada de noite na casa de um de vocês, ou jogando, ou bebendo (chá), ou comendo em um rodízio ou até mesmo não fazendo nada. Obrigado por nunca precisarem de hora ou lugar para conversar.

Gostaria também de agradecer a minha namorada pelo suporte e carinho oferecidos nessa parte final da minha graduação. Obrigado por me entender, me ajudar com meu péssimo português, com alguns trabalhos e até mesmo com a minha monografia, mesmo achando que ela é só "anotar o tempo que gasta colocar um arquivo no pen-drive".

Agradeço aos professores que desempenharam com dedicação as aulas ministradas.

Agradeço à minha orientadora, professora doutora Maristela, que com muita paciência e fôlego, conseguiu corrigir os meus textos e por ser uma excelente professora e profissional, a qual me espelho.

No mais, agradeço à todos que direta, ou indiretamente, fizeram parte da minha formação acadêmica.

Resumo

Projetos de Bioinformática, geralmente„ são executados como *workflows* científicos. Biólogos podem executar um mesmo *workflow* diversas vezes com diferentes parâmetros, com o objetivo de comparar os resultados obtidos e refinar a análise dos dados. Essas execuções geram vários arquivos com formatos e tamanhos diferentes, os quais precisam ser armazenados para futuras execuções. Para o gerenciamento de grandes volumes de dados, novos modelos de banco de dados, denominados NoSQL (*Not Only SQL*), tem sido especificados. Neste contexto, esta monografia apresenta uma avaliação do uso do sistema gerenciador de banco de dados Cassandra em *workflows* científicos de Bioinformática.

Palavras-chave: Bioinformática, Cassandra, NoSQL, Workflow.

Abstract

Projects in bioinformatics are usually executed as scientific workflows. Biologists often execute the same workflow many times with different settings so that data is refined. These executions generate many files with different size and formats which needs to be stored for further executions. To manage huge volumes of data, database models known as NoSQL (Not Only SQL) are being specified. In this context, this undergraduate thesis presents an evaluation of the Cassandra database in bioinformatic scientifics workflows.

Keywords: Bioinformatics, Cassandra, NoSQL, Workflow.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Bioinformática surgiu pela necessidade de ferramentas computacionais para a análise de dados genômicos originados pelos sequenciadores dos projetos genoma. Com os crescentes estudos nessa área surgiu uma grande demanda por uma forma de sequenciamento de baixo custo, estimulando o desenvolvimento de tecnologias de sequenciamento paralelos, produzindo milhões de sequências em uma única execução [22]. O baixo custo proporcionou um aumento na quantidade de dados gerados em projetos de Bioinformática.

Projetos de Bioinformática podem ser executados como *workflows* científicos, que são experimentos baseados em simulações computacionais, compostos por uma sequência de programas, cada qual com um conjunto de parâmetros e dados de entrada especificados para resolver um problema [40].

Biólogos podem executar o mesmo *workflow* de Bioinformática diversas vezes com diferentes parâmetros, a fim de comparar os resultados obtidos em cada execução e refinar a análise dos dados. Cada execução de um *workflow* científico de Bioinformática pode gerar um grande volume de dados e o armazenamento destes dados gera um desafio. Novos modelos de banco de dados, denominados NoSQL (*Not Only SQL*), tem sido especificados para tratar do gerenciamento de grande volume de dados.

Os bancos de dados NoSQL surgiram da necessidade de uma alta escalabilidade e de uma performance superior quando o volume de dados é muito grande, sendo aplicados em armazenamento distribuído de dados e projetados para suportar o acesso a dados em grande escala. Tratar da escrita e da leitura de grandes quantidades de dados pode ser um desafio para bancos de dados relacionais, sendo necessário, algumas vezes, aumentar a capacidade do servidor com mais memória e disco.

O SGBD (Sistema Gerenciador de Banco de Dados) Cassandra é uma das aplicações NoSQL mais utilizadas, segundo *Holt et al.* [58]. Por esse motivo foi escolhido para o desenvolvimento deste trabalho.

O Cassandra é uma implementação da família de colunas do modelo de dados Big Table [21], da Google, e usa aspectos de arquitetura *peer-to-peer* introduzido pelo Amazon Dynamo [23]. Dentre suas características estão presentes: a alta escalabilidade e a disponibilidade, sem um ponto único de falha e o alto desempenho em operações de leitura. Uma configuração do Cassandra bastante explorada é o fator de replicação, que descreve quantas cópias do dado inserido existirá dentro do *cluster* no qual se encontra. A variação dessa configuração pode ser utilizada a fim de garantir uma melhor disponibilidade dos dados.

Neste contexto, esta monografia apresenta uma análise do comportamento do sistema gerenciador de banco de dados NoSQL Cassandra em relação ao tempo de armazenamento e recuperação em diferentes formatos de dados biológicos, baseados no fator de replicação.

1.1 Objetivo

O objetivo desta monografia é realizar uma avaliação do comportamento do sistema gerenciador de banco de dados NoSQL Cassandra com dados de *workflow* em bioinformática, possuindo foco maior no impacto da configuração dos fatores de replicação deste sistema gerencador de banco de dados.

1.1.1 Objetivos Específicos

Para atingir o objetivo geral, os seguintes objetivos específicos foram definidos:

- Modelar um banco de dados para dados biológicos no NoSQL Cassandra;
- Especificar o processo para inserção e extração de dados biológicos no banco de dados NoSQL Cassandra;
- Realizar testes variando o número de máquinas e o fator de replicação no Cassandra.
- Avaliar o comportamento do sistema gerenciador de banco de dados Cassandra nos ambientes de teste.

1.2 Estrutura do Trabalho

O trabalho está estruturado nos capítulos a seguir:

- O Capítulo 2 apresenta o referencial teórico necessário para entender o tema de Bioinformática e de banco de dados, mostrando o que é um *workflow*, quais os tipos de dados da Bioinformática, envolvendo a parte de banco de dados relacionais e banco de dados NoSQL;

- O Capítulo 3 apresenta o referencial teórico referente ao banco de dados NoSQL Cassandra, suas características, modelo de dados e arquitetura;
- O Capítulo 4 especifica o estudo de caso aplicado para realizar os testes de inserção e extração no banco de dados NoSQL Cassandra, assim como os resultados obtidos e a análise desses dados;
- Por fim, no Capítulo 5, são apresentadas as conclusões e os trabalhos futuros.

Capítulo 2

Bioinformática e Banco de Dados

Tradicionalmente, quando os bancos de dados são utilizados para armazenamento de dados da Bioinfomática, eles são implementados utilizando o modelo relacional, no qual os dados são armazenados em tabelas. Com os novos desafios oriundo, principalmente, pelo grande volume de dados, novos modelos de banco de dados, chamados NoSQL estão sendo especificados. Neste capítulo são apresentados os conceitos fundamentais referentes à Bioinformática, banco de dados relacional e banco de dados NoSQL. A Seção 2.1 introduz os conceitos acerca da Bioinformática. Já a Seção 2.1.1 descreve os conceitos sobre *workflows* científicos de Bioinformática, como ele é estruturado e suas fases. A Seção 2.1.2 descreve os diferentes tipos de dados em *workflows* de Bioinformática. A Seção 2.2 apresenta uma introdução aos bancos relacionais, e por fim a Seção 2.3 trata sobre bancos de dados NoSQL.

2.1 Bioinformática

A Bioinformática tem como objetivo realizar análises de dados biológicos, como sequências de bases de DNA e genes, assim como predizer a estrutura e função de diversas macromoléculas [13]. Além das áreas de biologia molecular e ciência da computação, a área da matemática e da estatística estão presentes, norteando a Bioinformática.

Com o crescente avanço na área de biologia molecular e a necessidade de automação do mapeamento de sequências genéticas por diversos projetos, como o projeto Genoma Humano [41] entre os anos de 1980 e 1990, o crescimento da área, desde então, foi exponencial. Em termos de processamento, enquanto na metade dos anos 70 o sequenciamento de 150 nucleotídios levava cerca de dois meses, no anos 2000, o sequenciamento passou da ordem de milhões de nucleotídios por dia [11].

Além do aumento do sequenciamento dos nucleotídios, com a evolução da tecnologia, o preço para o sequenciamento genômico tornou-se menor. Através do gráfico da Figura 2.1

observa-se que o preço para sequenciar o genoma humano diminui ao decorrer dos anos passando de cem milhões, em 2001, para apenas mil dólares em 2015. A diminuição do custo é graças a aparição da tecnologias chamadas de *Next-Generation Sequencing* (NGS), como por exemplo, o sistema Illumina HiSeq X Ten [5] que foi o responsável por atingir o marco de mil dólares para sequenciar todo o genoma humano [20].

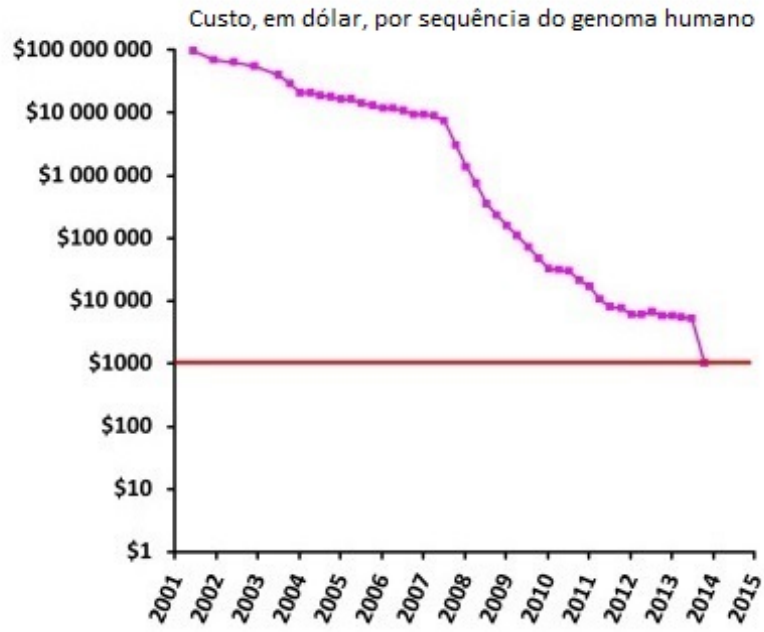

Figura 2.1: Gráfico do Custo pelo Sequenciamento do Genoma Humano, adaptado de [36].

2.1.1 Workflow Científico

Experimentos realizados na Bioinformática, geralmente, são modelados como *workflows* científicos. *Workflows* científicos de Bioinformática são relacionados ao processamento de dados biológicos, especialmente com sequenciamento de DNA e/ou RNA, e podem ter diferentes processos envolvidos. Comumente, como apresentado na Figura 2.2, estes *wokflows* tem três fases principais [48]: filtragem, mapeamento/montagem e análise/anotação.

Uma característica sobre a sequência das fases é que a saída da fase anterior é a entrada da fase posterior, ou seja, assim que a primeira fase executar, sua saída será a entrada para a segunda fase, e assim por diante [28].

Antes que todo o processo do *workflow* comece, os biólogos fazem a coleta das amostras de material biológico (DNA ou RNA). Em seguida, esse material é processado por um sequenciador de alto desempenho, como o Illumina [53], e a partir desse processo, se

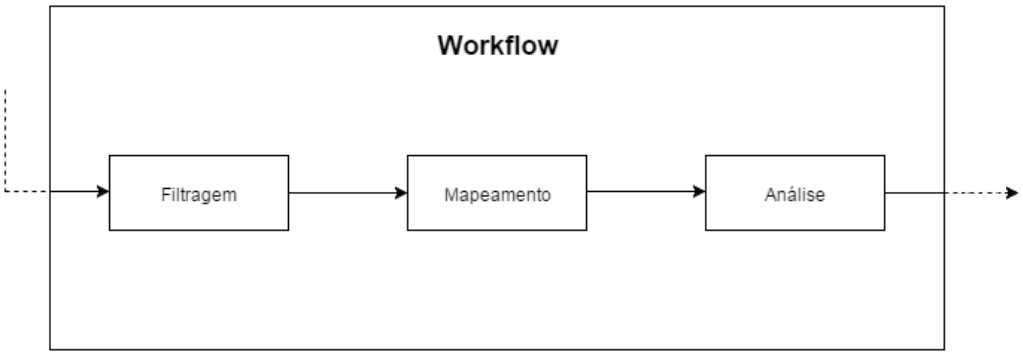

Figura 2.2: Exemplo de *Workflow* com Três Fases..

obtém as chamadas *reads*, que são um conjunto de diversas bases nitrogenadas formando pequenos fragmentos/porções do DNA ou RNA já sequenciado.

O estágio de filtragem é responsável por receber como entrada um arquivo e filtrá-lo de modo a remover algum fragmento que se julgue desnecessário ou não confiável. As *reads* são geradas pelo sequenciador genético em um arquivo com o formato FASTA. Quando o arquivo é gerado cada uma das bases nitrogenadas identificadas possuem um fator de qualidade associado. Esse fator de qualidade determina a probabilidade de erro da base durante a identificação pelo sequenciador.

Além da possibilidade de querer eliminar as bases nitrogenadas com alta taxa de erro, pode-se desejar também, remover regiões que foram sequenciadas que não interessam ou que possam dificultar o processo das etapas posteriores no *workflow* [7]. O modo como a filtragem se realiza pode variar entre pequenos *scritps* gerados por um usuário até programa mais complexo com diversas opções de uso. Um programa em destaque é o FASTX-Toolkit que é uma coleção de ferramentas de linha de comando para o processamento de arquivos FASTA/FASTQ [25].

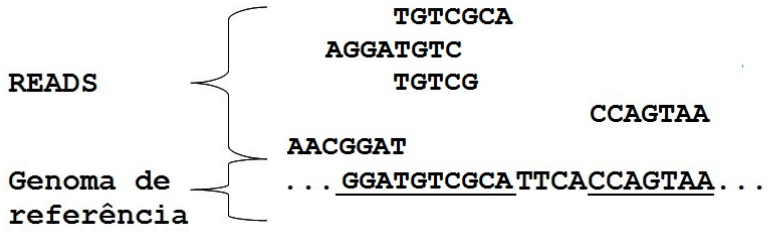

Figura 2.3: Exemplo de Mapeamento.

Quando não é conhecido o genoma de referência, pode-se alinhar as *reads* entre elas gerando sequências maiores chamadas *contigs*. A esse processo dá-se o nome de montagem.

Na Figura 2.4 diversas *reads* são alinhadas e a sombra dessas *reads* mostra qual é a porção do cromossomo sequenciado.

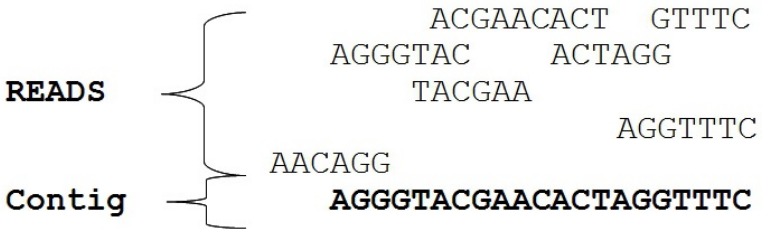

Figura 2.4: Exemplo de Montagem.

Quando se trata de mapeamento de RNA, a dificuldade está na existência de *splice-junctions* nas *reads* sequenciadas. As Figuras 2.5 e 2.6 mostram o problema encontrado no processo de alinhamento de sequências de transcritos com *splice-junctions*. Na Figura 2.5 tem-se o processo de transcrição, no qual há o descarte do *intron* e somente os *exons* são copiados para formar o transcrito a partir do genoma de referência.

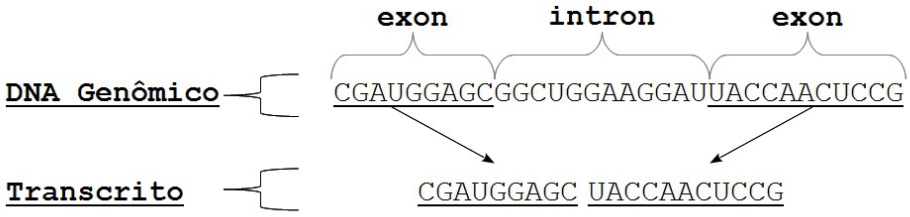

Figura 2.5: Exemplo de *read* sem *splice-junction*.

Na Figura 2.6, quando a *read* com o transcrito é alinhada contra o genoma de referência, o *intron* do genoma de referência permite a existência de dois possíveis alinhamentos para a *read*, fazendo com que a *read* não encontre sua posição adequadamente. Quando tal fato ocorre, muitas *reads* podem ser descartadas por não encontrarem um alinhamento adequado.

A fase de mapeamento para realizar sua tarefa exige algoritmos bem mais complexos. Vários programas podem ser encontrados para que se possa reduzir o tempo de execução nesta fase, cada um oferecendo diferentes funcionalidades. Em destaque, o programa TopHat [9] é uma boa opção devido sua facilidade de uso e o algoritmo eficiente para tratamento de memória com *reads* menores. O TopHat, que é responsável por fazer o tratamento das *splice-junctions* usa como base um outro programa, o Bowtie [4], que é o responsável pelo alinhamento das *short-reads*, utilizando mais de um processador durante

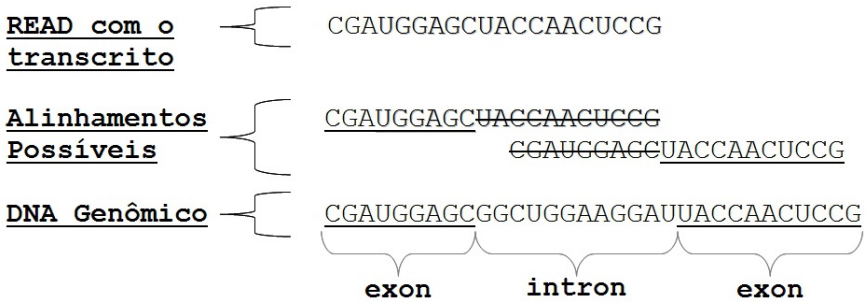

Figura 2.6: Exemplo de *read* com *splice-junction*.

a tarefa. Os dois programas combinados realizam a operação de mapeamento com uma grande eficiência e desempenho.

A fase final de um *workflow* é a de análise. Nesta fase, procura-se fazer a interpretação e a análise dos dados brutos obtidos pela fase anterior, também deve-se levar em consideração o objetivo do experimento e o que está sendo analisado para que se possa inferir algum tipo de informação. Caso seja necessário fazer a análise para sequências genéticas que são suscetíveis ou não a um tipo de vírus, primeiramente deve-se faz o mapeamento de várias amostras de RNA de indivíduos que são suscetíveis, em seguida realiza-se o mesmo processo para os que não são suscetíveis, e por último deve-se fazer a análise, comparando os dois resultados do mapeamento, buscando saber quais são os trechos de DNA que possam estar dando a imunidade aos indivíduos não suscetíveis ou possam estar causando a deficiência aos indivíduos suscetíveis.

Os arquivos gerados pela fase de mapeamento são grandes o suficiente para inviabilizar a análise manual, por esse motivo os programas estatísticos são bons auxiliadores. Um exemplo de tal programa é o R [47], que é uma linguagem para processamento e gráficos estatísticos com diversas ferramentas para análises estatísticas. Outro programa é o *Basic Local Alignment Search Tool* (BLAST) [8], que tem como principal funcionalidade descobrir a existência de um gene em um genoma, e qual a sua funcionalidade.

2.1.2 Dados da Bioinformática

Assim como existem diversos programas que podem ser usados em cada uma das fases do processo de um *workflow* em Bioinformática, também existem diferentes formatos de arquivos de entrada e saídas. O tipo de arquivo depende do programa utilizado e da necessidade de cada fase. No projeto apresentando por este trabalho, o arquivo que contém os dados a serem inseridos para a primeira fase do *workflow*, a fase de filtragem, são arquivos biológicos no formato FASTQ e FASTA [44].

Os formatos FASTQ e FASTA são bastante utilizados em processos de sequenciamento e na maior parte dos casos os arquivos gerados nos sequenciadores são convertidos para estes formatos, que são aceitos pela maioria dos programas próprios para filtragem ou mapeamento das *reads*.

O formato FASTQ trata-se de um arquivo de texto que serve para armazenar uma sequência biológica, geralmente, sendo uma sequência de nucleotídios, e seus índices de qualidade correspondentes. Os dados são codificados usando caracteres ASCII para serem abreviados [53]. A Figura 2.7 mostra, como exemplo, um trecho de um arquivo FASTQ que foi utilizado neste trabalho. A diferença entre o arquivo FASTQ e FASTA é que no FASTA as *reads* e os índices de qualidade estão em arquivos separados, sendo que os índices de qualidade são opcionais e o arquivo pode não existir.

```
@SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=72
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACCAAGTTACCCTTAACAACTTAAGGGTTTTCAAATAGA
+SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=72
IIIIIIIIIIIIIIIIIIIIIIIIIII9IG9ICIIIIIIIIIIIIIIIIIIDIIIIII>IIIII/
@SRR001666.2 071112_SLXA-EAS1_s_7:5:1:801:338 length=72
GTTCAGGGATACGACGTTTGTATTTTAAGAATCTGAAGCAGAAGTCGATGATAATACGCGTCGTTTTATCAT
+SRR001666.2 071112_SLXA-EAS1_s_7:5:1:801:338 length=72
IIIIIIIIIIIIIIIIIIIIIIIIIIII6IBIIIIIIIIIIIIIIIIIIIIGII>IIII-I)8I
```

Figura 2.7: Exemplo de um Arquivo FASTQ.

O arquivo FASTQ é divido em vários blocos, que recebem o nome de *short-reads sequence* (SRS). O bloco de uma SRS é composto por quatro linhas, sendo que a primeira linha é o identificador da sequência, a segunda é a sequência em si, representada pelas letras de sua base nitrogenada, a terceira é o identificador dos índices de qualidade e a quarta são os identificadores respectivos para cada base. A Figura 2.7 apresenta um exemplo de um arquivo FASTQ e são mostradas duas SRS, no qual a primeira SRS é iniciada na primeira linha pelo identificador "@" e termina ao final da quarta linha. Logo em seguida, no início da quinta linha, pode-se observar novamente o identificador da próxima SRS.

Outros formatos são necessários para representar alinhamentos de sequências de DNA ou RNA, tais como o *Sequence Alignment/Map* (SAM) e *Binary Alignment/Map* (BAM) [26], que são utilizados para a fase de mapeamento. Ambos os formatos armazenam as mesmas informações, sendo que o formato SAM é gravado no formato texto, enquanto que o BAM é codificado no formato binário, sendo comprimido a fim de ocupar menos espaço em disco.

A Figura 2.8 mostra um exemplo de um arquivo no formato SAM. As duas primeiras linhas do arquivo, iniciadas pelo caractere "@", são os cabeçalhos com informações gerais sobre os alinhamentos, tais como o nome da sequência de referência, seu tamanho e a

ordem. As demais linhas trazem informações sobre cada alinhamento encontrado. As sequências que foram alinhadas estão sublinhadas.

```
@HD VN:1.3 SO:coordinate
@SQ SN:ref LN:45
r001 163 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002   0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA     *
r003   0 ref  9 30 5H6M       *  0   0 AGCTAA    *   NM:i:1
r004   0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC        *
r003  16 ref 29 30 6H5M       *  0   0 TAGGC     *   NM:i:0
r001  83 ref 37 30 9M         =  7 -39 CAGCGCCAT          *
```

Figura 2.8: Exemplo de um Arquivo SAM.

Por fim, para a fase de análise, os formatos de arquivos são diversos, pois a análise depende do objetivo que se deseja encontrar, portanto, os dados gerados podem ter diferentes formatos. Por exemplo, o programa R, que é bastante utilizado para análises estatísticas, possui a opção que permite o usuário exportar os resultados em diversos formatos, inclusive formato texto, indicando quais campos deseja-se exportar.

2.2 Banco de Dados Relacionais

Segundo Naphtali Rishe [42], um sistema de banco de dados pode ser definido como "o armazenamento de dados atualizáveis de uma aplicação e um software de gestão que oculta do usuário o aspecto físico do armazenamento da informação e a representação da informação" (tradução livre). Em outras palavras, um sistema de banco de dados pode ser a junção de um banco de dados, uma coleção de dados organizados e persistentes, e um sistema gerenciador de banco de dados (SGBD), uma aplicação que possui responsabilidade por gerenciar esses dados e controlar toda a estrutura.

Um banco de dados relacional é composto por tabelas e pela forma como essas tabelas interagem entre si. Em uma visão mais externa, tem-se que o modelo, ou seja, a estruturação e a descrição dos dados é conhecida como esquema. O esquema por sua vez tem diversas representações de modelos de dados que são as tabelas. As tabelas são compostas por metadados estruturados, chamados de colunas, que possuem um nome e um tipo associado [50]. A Figura 2.9 mostra como é a estruturação de uma tabela que faz a representação de um aluno. A tabela possui os metadados do nome do aluno, sua matricula, idade e seu IRA (Índice de Rendimento Acadêmico). A cada metadado existe uma tipagem associada, *int* é a representação de números inteiros, *string* é a representação de caracteres alfa-numéricos e *float* é a representação de números com casa decimais.

Um registro é uma instância da tabela, ou seja, a linha que contém os dados referentes a cada metadado. A Tabela 2.1 possui 4 registros com dados de alunos de acordo com os

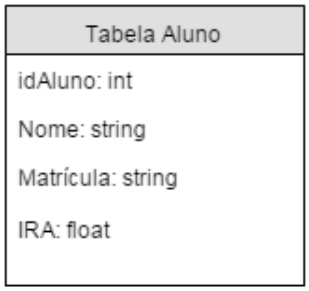

Figura 2.9: Representação dos Metadados da Tabela Aluno.

Tabela 2.1: Exemplo da Tabela Aluno Populada.

idAluno	**Nome**	**Matrícula**	**IRA**
1	Ricardo	09/0115447	3,23
2	Isabela	10/0144246	4,2
3	Emily	13/0084255	2,92
4	Thomas	13/0121123	3,55

metadados referentes a cada uma das colunas. Cada linha na tabela é um registro que descreve uma entidade aluno. Pela tabela, o aluno com o dado, referente ao metadado idAluno 1, possui nome Ricardo, matrícula 09/0115447 e IRA 3,23.

A linguagem de comunicação entre a aplicação e o usuário de um banco de dados relacional é feita através de SQL (*Structured Query Language*). O SQL é uma linguagem de *script* declarativa, simples de se utilizar, baseada na álgebra relacional. Muito utilizada para realizar diversos tipos de ações na utilização de banco dados, sendo as operações de CRUD (*Create*, *Read*, *Update*, *Delete*), as mais comuns.

No modelo relacional existem dois conceitos que são fundamentais para que se possa ter um bom funcionamento. O primeiro deles é o conceito de atributos chave, sendo ela primária ou estrangeira, e o segundo é o de restrição de integridade.

A chave primária é um atributo, ou um conjunto de atributos, que garante unicidade de cada registro (linha) em uma tabela. Essa abordagem é necessária para que se possa localizar qualquer registro da tabela de forma que não haja nenhum tipo de ambiguidade [6]. A chave estrangeira, por sua vez, funciona como uma referência de uma chave primária para uma outra tabela. Ela é utilizada para implementar relacionamentos entre tabelas no banco de dados.

A definição do conceito de restrição de integridade (RI) dada por Carlos Heuser [6] é que uma RI é uma condição especificada sobre o esquema de um banco de dados que limita os dados que possam ser armazenados em uma de suas instância. Os sistemas gerenciadores de banco de dados impõe restrições de integridade, significando que eles só

permitem que instâncias que obedecem as restrições de integridade possam ser armazenadas no banco de dados.

2.2.1 Propriedades ACID

Uma outra importante função de sistemas gerenciadores de banco de dados é o gerenciamento de transações. Uma transação pode ser definida como uma coleção de operações que desempenham uma função lógica dentro de uma aplicação do sistema de banco de dados. Existem algumas propriedades para garantir a consistência dos dados e o funcionamento das execuções das transações de maneira correta. Tais propriedades são: Atomicidade, Consistência, Isolamento e Durabilidade, mais conhecida como ACID [31].

- Atomicidade: As operações sempre devem ser executadas, de forma que a transação ou é executada por completo, ou nada será executado;
- Consistência: Assim que uma operação for concluída, o banco de dados deve manter seu estado de consistência, ou seja, deve satisfazer as restrições de integridade;
- Isolamento: Caso duas operações estejam sendo realizadas paralelamente, uma não deve interferir na outra e cada operação deve ter seus efeitos isolados;
- Durabilidade: Caso uma operação seja concluída com sucesso, seu efeito não poderá ser mais desfeito.

2.3 Banco de Dados NoSQL

Novos modelos de banco de dados tem sido definidos, sendo chamados de NoSQL (*Not Only SQL*) que significa "não apenas SQL". Este termo se refere aos sistemas gerenciadores de banco de dados que não adotam o modelo relacional e são mais flexíveis em relação às propriedades ACID. Esta adaptabilidade é relevante, pois é necessário ter uma alta escalabilidade para gerenciar grandes quantidades de dados, bem como para garantir a alta disponibilidade desses dados.

2.3.1 Características dos Bancos de Dados NoSQL

Dentre as principais características dos bancos de dados NoSQL estão a desnormalização das informações, e alto desempenho no armazenamento paralelo de dados em grande escala. Eles apresentam a capacidade de possuírem escalabilidade horizontal, esquema flexível, baixo tempo de resposta e grande disponibilidade de acesso.

Quanto ao tipo de controle de consistência, os bancos de dados NoSQL implementam o modelo conhecido como BASE (*Basically Available*, *Soft-state*, *Eventually consistency*), que se opõe ao *ACID* no modelo relacional. Seus itens são descrito da seguinte forma [12]:

- Basicamente disponível: O banco de dados NoSQL apresenta disponibilidade de dados até na presença de múltiplas falhas. Esse método é atingido usando uma abordagem de distribuição em larga escala, em vez de manter um único servidor de dados que prioriza a tolerância a falhas, o dado é divido em diversos sistemas de armazenamento com um grande nível de replicação. Caso ocorra um evento de falha no acesso ao dado, não significa que toda a base será compromissada ou não estará disponível;
- Estado leve: Os bancos de dados que implementam o BASE abandonam totalmente os requerimentos de consistência que o modelo ACID implementa. Um dos conceitos básicos por trás do BASE é que a consistência dos dados é responsabilidade do desenvolvedor e não deve ser tratado pelo banco de dados;
- Consistência eventual: A única exigência que os bancos de dados NoSQL possui em relação à consistência é que em algum ponto no futuro os dados irão convergir para um estado consistente, porém não existe garantias ou um tempo definido para quando isso deve ocorrer. Isso é uma oposição a necessidade de consistência imediata que é requerida pelo ACID, que proíbe uma transação de acontecer até o banco de dados tenham convergido para um estado consistente.

Existe um teorema criado por Eric Brewer nos anos 90, conhecido como o teorema CAP ou o teorema de Brewer, que quantifica *trade-offs* inevitáveis. Nesse teorema, Brewer cita a existência de três propriedades principais em sistemas distribuídos, são elas [17]:

- Consistência: propriedade que se refere ao sistema estar em uma versão consistente após a execução de uma determinada operação. Um sistema é dito consistente se uma consulta à base de dados por parte de vários usuários resulta em uma mesma visão após uma atualização feita por um outro usuário que possui permissão para escrever no sistema;
- Disponibilidade: propriedade que garante a disponibilidade do serviço a qualquer momento. Ou seja, qualquer usuário terá sua requisição atendida a todo momento, mesmo se uma máquina em um *cluster* estiver fora do ar;
- Tolerância à partição: propriedade em que um sistema pode executar corretamente, mantendo todas as suas características independente da divisão física da rede, de

perda de dados ou de falhas no sistema. Um sistema com alta tolerância ao particionamento é um sistema altamente escalável, pois possui a habilidade de lidar com a adição e remoção dinâmica de nós de recursos.

O teorema CAP aborda que, dentre as três propriedades, só é possível utilizar duas em um sistema distribuído. É importante enfatizar que segundo Brewer, os modelos ACID e BASE são dois extremos de um espectro em uma relação consistência-disponibilidade e que diversos sistemas modernos de larga escala usam um meio termo desse espectro [18].

A impossibilidade de se escolher mais de duas propriedades da-se ao fato de que ao escolher duas delas, a outra é inviabilizada. Em outras palavras, pode-se dizer que caso exista consistência e disponibilidade ao se dar liberdade para o usuário de ler e escrever a qualquer momento, sendo que todos os usuários estarão vendo uma versão consistente do sistema, não é possível permitir que a rede tenha uma divisão física, pois a divisão física quebraria o princípio da consistência.

Seguindo o raciocínio, se há disponibilidade e tolerância à partição, a consistência não é alcançada, pois um usuário poderia escrever e ler a qualquer momento, porém ao encontrar um evento de partição seria necessário esperar até que todo o dado voltasse a ser consistente e durante esse tempo o serviço não estaria disponível.

Na última possibilidade, caso um sistema possua consistência e tolerância à partição, não é possível garantir para o usuário a escrita e leitura a todo momento. Ao se dar essa liberdade de leitura e de escrita, o sistema distribuído não tem como se manter estável para todos que o vêem.

A Figura 2.10 mostra quais são as possíveis relações entre as propriedades do CAP e as classifica em três categorias: Consistência e Disponibilidade (CA), Consistência e Tolerância à partição (CP) e Disponibilidade e Tolerância à partição (AP). Em cada uma dessas categorias são mostrados alguns bancos de dados que as representam.

2.3.2 Modelo de Dados NoSQL

Enquanto no modelo relacional os sistemas de gerência do banco são realizados apenas com dados estruturados em tabelas, nos sistemas de gerenciadores dos bancos de dados NoSQL existem diferentes modelos de dados. Os modelos NoSQL são classificados como [10][49][57]:

Modelo chave-valor:

O modelo de armazenamento chave-valor pode ser definido como uma tabela *hash*, na qual existe uma chave única e cada uma das chaves estão associadas a um único valor. Esses valores podem ser simples *strings* de texto, assim como formas de dados mais complexas. Pela facilidade de implementação e pela unicidade de cada chave,

Figura 2.10: Possibilidades do Teorema CAP, e Exemplos de Banco de Dados.

o modelo chave-valor permite que os dados sejam acessados muito rapidamente pela chave, isso principalmente em sistemas que possuem alta escalabilidade, o que também contribui para o aumento da disponibilidade de acesso aos dados [49]. Um exemplo de banco de dados do modelo chave-valor é o Dynamo [23], usado pela Amazon.

Modelo orientado a documentos:

Em um modelo orientado a documentos os dados armazenados são coleções de documentos. Um documento pode ser descrito como um objeto que relaciona uma chave única a um conjunto de dados associados.

O modelo orientado a documentos se assemelha ao chave-valor, porém no chave-valor uma única tabela *hash* é criada para todo o banco de dados. Enquanto no orientado a documentos, existe um conjunto de documentos e em cada um desses documentos existe um conjunto de chaves com seus respectivos campos associados. Pode-se fazer uma analogia referente ao modelo orientado a documentos dizendo que tal modelo é um modelo chave-valor, no qual é possível fazer consulta dentro um campo.

Uma das características principais desse modelo é que um documento é identificado dentro de uma coleção através de um identificador e tal identificador deve ser único. Assim como a chave que identifica o conjunto de dados dentro do documento. Uma

outra característica importante sobre este modelo é que ele não depende de um esquema rígido, ou seja, não há exigência de uma estrutura fixa. Desse jeito é possível que uma atualização na estrutura do documento exista sem que ela cause problemas ao banco de dados [49]. Essa flexibilidade em atualizar a estrutura dos documentos é uma das principais vantagens do modelo orientado a documentos. Pode-se citar o CouchDB [30] e o MongoDB [37] como banco de dados baseados no modelo orientado a documentos.

Modelo orientado a colunas:

Primeiramente, deve-se entender que o modelo orientado a colunas segue um padrão diferente dos modelos citados anteriormente. O modelo orientado a colunas, como o próprio nome já demonstra, relaciona um conjunto de dados de forma tabular, ou seja, em uma coluna e não em uma linha. Neste tipo de modelo os dados são indexados por uma tripla (linha, coluna e *timestamp*), onde as linhas e as colunas são identificadas por chaves e o *timestamp* é o que permite identificar as diferentes versões de um mesmo dado.

Em um banco de dados orientado a colunas as operações de leitura e escrita são atômicas, ou seja, todos os valores associados a uma linha são considerados na execução da operação de leitura ou escrita. Um outro conceito importante sobre esse modelo é o de família de colunas, na qual as colunas podem ser organizadas em famílias de modo a facilitar a organização e o particionamento, podendo também ser adicionadas novas colunas a qualquer momento.

O ponto negativo da família de colunas é que elas não podem ser definidas durante a execução, resultando em uma menor flexibilidade quando comparada aos modelos previamente vistos [49]. O modelo orientado a colunas tem como principal banco de dados o BigTable [21] da Google, o Apache HBase [38] e o Cassandra [19].

Modelo orientado a grafos:

O modelo orientado a grafos é especializado na gerência eficiente de dados que estão conectados entre si. Pode-se descrever o modelo orientado a grafos com três componentes básicos [49]: os nós (que são conhecidos como os vértices do grafo), os relacionamentos entre os nós (que são conhecidos como os vértices do grafo) e a propriedades (que podem ser chamados atributos) dos nós e dos relacionamentos.

Neste modelo cada nó pode ser conectado por mais de uma aresta, o que se compara a uma estrutura de multigrafo direcionado. Devido a essa característica pode-se atribuir tal modelo de banco de dados às aplicações baseadas em dados que possuam muitas relações, pois o custo de buscas com muitas conexões, se comparado com o modelo relacional, é elevado [10]. Dessa maneira esse tipo de consulta teria uma

ganho de performance, o que permitiria um melhor desempenho das aplicações. Exemplos de banco de dados baseados em grafos são: Neo4j [35], AllegroGraph [29] e Virtuoso [43].

2.3.3 Considerações sobre o Modelo Relacional e NoSQL

A construção de um banco de dados seguindo o modelo relacional pode gerar um grande número de tabelas que se relacionam através de suas chaves. Quanto maior o modelo, maior sua complexidade e maior o número de dados que serão armazenados no banco de dados.

A Figura 2.11 demonstra a escalabilidade de banco de dados relacionais contra banco de dados NoSQL. Nota-se que quanto menor o número de dados, o modelo relacional possui uma melhor performance em relação ao modelo NoSQL, porém com o aumento do volume de dados, o modelo relacional começa a decair e o NoSQL começa a ter uma performance melhor.

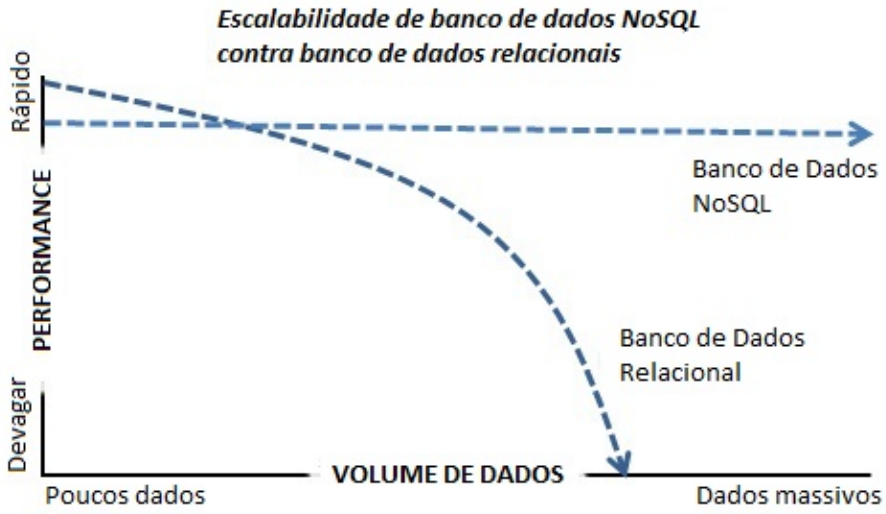

Figura 2.11: Escalabilidade do SQL x NoSQL, adaptado de [24].

Em uma análise comparativa entre banco de dados relacionais e NoSQL feita por B. Cooper et al. [2] e R. W. Brito [52], o desempenho dos bancos de dados relacionais não é satisfatório devido ao crescimento do tempo de resposta de algumas consultas.

2.4 Trabalhos Relacionados

Armazenar dados biológicos não é um desafio recente, tendo em vista as dificuldades de gerenciamento dos dados como relatado em [54]. Em [56] e [46], foram propostos

modelos utilizando sistemas gerenciadores de banco de dados relacional, SQL Server 2008 e PostgreSQL, respectivamente, para armazenar dados genômicos.

Outros trabalhos utilizaram o modelo relacional Chado para armazenar esse tipo de dados. Em [33] foi desenvolvido um sistema para gerenciar *workflows* para a análise de dados genômicos, que suporta processamento de alto desempenho. Em [55] criaram-se uma nova versão para uma ferramenta de visualização e anotação de sequências de dados genômicos, a Artemis. Essa nova versão passou a ser capaz de ler e escrever diretamente no banco de dados Chado.

Em [57] é realizado um estudo da utilização do MongoDB para armazenar e extrair dados da fase de filtragem de um *workflow* científico de Bioinformática, enquanto no trabalho de [45] é realizado um estudo com os mesmo dados utilizando o Cassandra. Em [59] foi utilizado o Cassandra com múltiplos nós para garantir a consistência dos dados.

Diferentemente desdes artigos apresentados, esta monografia utiliza uma variedade dos formatos de arquivos armazenados e recuperados do sistema gerenciador de banco de dados, assim como uma variação no fator de replicação utilizado no Cassandra. Enquanto trabalhos realizados anteriormente tiveram foco em poucos formatos de arquivos, principalmente FASTA e FASTQ, neste foi utilizado diferentes fases do *workflow* e seus respectivos de formatos de arquivos, além do estudo de diferentes configurações computacionais para o ambiente.

Capítulo 3

NoSQL Cassandra

Esta monografia apresenta uma análise do comportamento do banco de dados NoSQL Cassandra no contexto de *workflow* de Bioinformática. Neste capítulo as características do Cassandra são apresentadas. A Seção 3.1 descreve a definição desse banco de dados, assim como suas características, modelo de dados e funcionalidade. A Seção 3.2 descreve sua arquitetura.

3.1 Definição

O Cassandra é um sistema gerenciador de banco de dados distribuído massivamente escalável, criado para armazenar uma grande quantidade de dados espalhados por várias máquinas e oferecer alta viabilidade de acesso e dados consistentes [1].

Originalmente criado para o Facebook, o Cassandra foi projetado para ter nós simétricos *peer-to-peer*, em vez de possuir nós centralizados, a fim de garantir a não existência de um único ponto de falha. O Cassandra automaticamente particiona os dados através de todos os nós no *cluster*, porém o administrador do banco de dados pode determinar qual é o dado que será replicado e quantas cópias do dado serão criadas.

Em 2008, o Facebook tornou o Casandra código aberto e em 2009 o Apache incubou o projeto. A partir de então o Cassandra está sendo usado por diversas empresas tais como: Netflix, Twitter, Instagram e Ebay. A Figura 3.1 é um demonstrativo ilustrativo da adoção do Cassandra pelas empresas e suas versões [15]. Como pode ser observado, na primeira versão (0.1), em julho de 2008, o Facebook era o único a utilizar o Cassandra, já entre as versões 0.7 e 1.0, o Twitter também começou a utilizar o Cassandra.

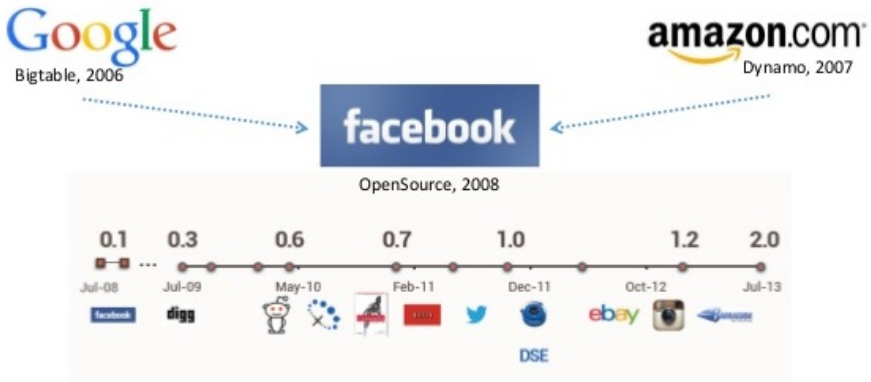

Figura 3.1: História do Cassandra, adaptado de [15].

3.1.1 Características

O Cassandra é uma implementação mista que junta a família de colunas do modelo de dados Big Table e aspectos de arquitetura *peer-to-peer* introduzidos pelo Dynamo. Alguns dos pontos positivos do Cassandra são [32]:

- Escalabilidade e disponibilidade, sem um ponto único de falha;
- Rendimento de escrita muito alto e bom rendimento de leitura;
- Linguagem de consulta semelhante a SQL;
- Esquema flexível.

As características principais do Cassandra, segundo Eben Hewiit [19], podem ser listadas como sendo um banco de dados: distribuído, descentralizado, escalável, com alta disponibilidade, tolerante a falhas, consistência e de alta performance. Uma descrição mais detalhada de cada característica é apresentada a seguir.

Distribuído e descentralizado

O Cassandra é capaz de executar em diversas máquinas, o que o torna distribuído, e por mais que ele esteja executando em diversas máquinas, para a visão usuário é como se ele estivesse executando em apenas uma. Sua implementação foi otimizada justamente para executar em diversas máquinas, sendo que um único *cluster* Cassandra pode executar em *data-centers* dispersos através do globo.

O fato de que o Cassandra não possui um único ponto de falha, o classifica como sendo descentralizado. Todos os nós do *cluster* funcionam exatamente do mesmo modo. A tal fato da-se o nome de simetria do servidor, ou seja, pode-se descrever como uma arquitetura *peer-to-peer* (uma topologia de anel) [19]. Uma vez que

todos os nós estão fazendo a mesma coisa, não existe um nó que coordena todas as atividades, ou seja, não existe uma relação mestre-escravo, gerando uma alta disponibilidade do sistema.

Escalabilidade Elástica

Segundo Hewiit [19] escalabilidade é "uma característica arquitetural de um sistema que pode continuar servindo um grande número de requisição com uma pequena degradação na performance"(tradução livre). Desse modo, uma abordagem de escalabilidade vertical é simplesmente aumentar o poder computacional, melhorar o hardware. Já uma abordagem de escalabilidade horizontal significa aumentar o número de máquinas e dividir os dados entre elas para que uma só máquina não tenha que ficar encarregada de atender todas as requisições.

A escalabilidade elástica refere-se à uma relação com a escalabilidade horizontal, no qual é possível aumentar e diminuir o número de máquinas dinamicamente. Dessa forma pode-se dizer que o *cluster* Cassandra estará sempre apto a receber máquinas e fazer a distribuição das cópias dos dados entre elas para atender às novas requisições, sem que seja necessário reiniciar o sistema, processo, ou até mesmo editar algum comando. De forma análoga também pode-se remover máquinas do *cluster* sem que ele pare de funcionar.

Disponibilidade e Tolerância a Falhas

Em termos arquiteturais, pode-se dizer que a disponibilidade de um sistema é medida de acordo com sua capacidade de completar requisições. Porém todo e qualquer computador pode sofrer de algum tipo de falha, que envolvem desde falhas nos componentes de hardware, erros de dados corrompidos e até mesmo erros com a rede.

O Cassandra busca reduzir as chances da ocorrência desses erros fazendo uso de redundâncias e realocação de recursos quando apresentam chances de falha. Porém, mesmo se ocorrer um erro, ele irá dar continuidade a um procedimento afim de obter um resultado final, apesar de estar parcialmente correto. Essa é uma medida para evitar que todo um projeto se comprometa devido a pequenos erros.

Consistência

Consistência essencialmente significa que uma operação de leitura sempre retornará o valor do dado correspondente em sua versão mais recente. Essa característica, quando se fala de sistemas altamente distribuídos, como o Cassandra, é bem difícil de se conseguir.

Cassandra troca uma parte de sua consistência para atingir um nível de disponibilidade total, por isso é definido como um banco de dados de "consistência eventual".

Segundo Hewiit [19], o Cassandra é um banco de dados de consistência ajustável, significando que é dado ao usuário a permissão de decidir qual é o nível de consistência desejada para a aplicação.

Alta Performance

O banco de dados Cassandra foi especificamente desenvolvido para tirar vantagem do uso de multiprocessamento, seja em máquinas que possuam multicores ou até mesmo em *data-centers* com diversas máquinas executando em conjunto. Ele pode escalar por centenas de terabytes mantendo a consistência. Seu uso é recomendado em ambientes que fazem uso dessas variações e precisam de alta performance para poder ter acessos rápidos às informações.

3.1.2 Modelo de Dados

Para se entender os modelos de dados, precisa-se primeiro entender duas estruturas básicas do Cassandra. A primeira estrutura básica são as colunas, um par chave-valor, similar ao modelo chave-valor, visto anteriormente. A Figura 3.2 exemplifica três pares chave-valor, no qual as chaves 1, 2 e 3 estão relacionadas com os valores 1, 2 e 3 respectivamente.

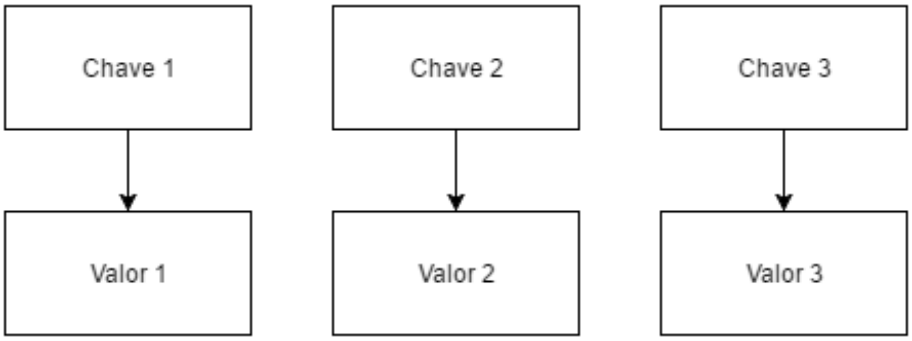

Figura 3.2: Exemplo de Coluna do Cassandra.

A outra estrutura básica é a família de colunas, que é definida como uma divisão lógica que associam dados similares [19], em outras palavras, é um contêiner para linhas que possuem colunas similares, mas não necessariamente as mesmas. Uma família de colunas é descrita pela Figura 3.3. A figura apresenta duas famílias de colunas, tomando como exemplo a *row key* 1, pode-se dizer que ela é a chave que representa a família de colunas representadas pelo contêiner com as colunas 1, 2 e 3.

Seguindo uma linha do tempo de versões do Cassandra, pode-se dizer que o modelo de dados inicial era composto por quatro estruturas[19]: as super colunas, a família de colunas, as colunas e as linhas. Entretanto, no decorrer de suas distribuições, o modelo de dados foi sofrendo diversas alterações. Em sua primeira versão, cada família de colunas

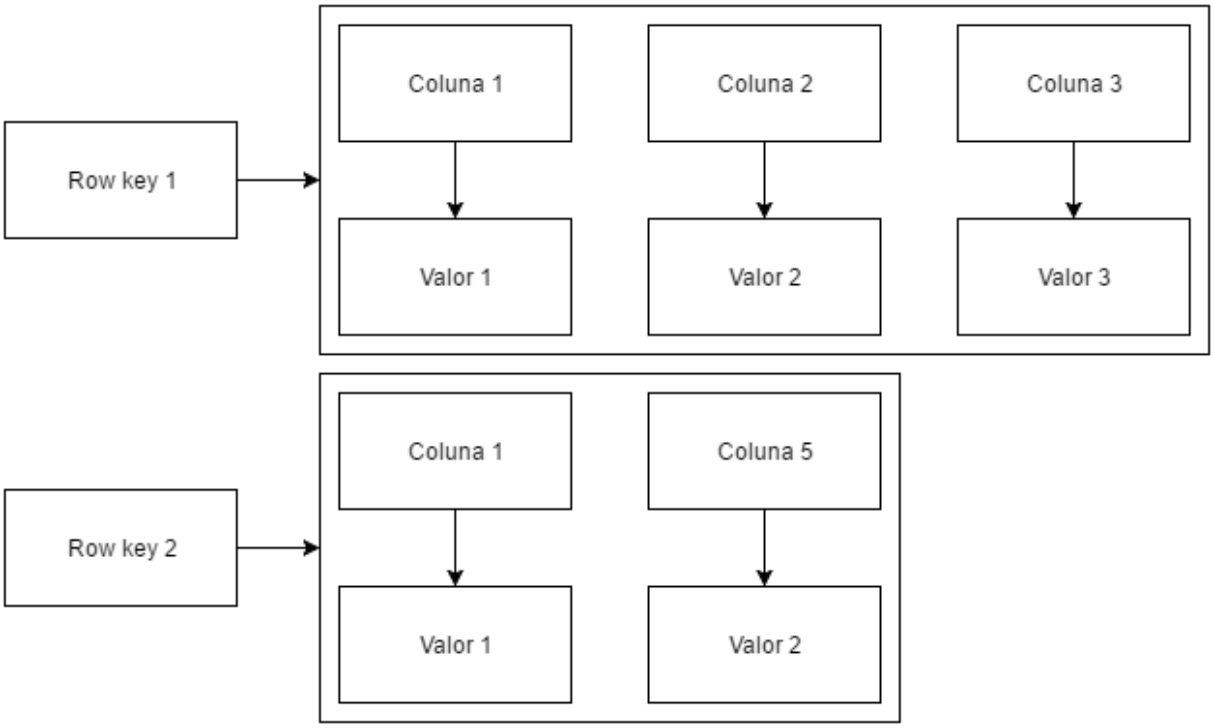

Figura 3.3: Exemplo de Família de Coluna do Cassandra.

possuía um conjunto de colunas, já as super colunas continham um conjunto de colunas ou um conjunto de famílias de colunas [21].

Na versão 1.1 do Cassandra, as super colunas foram descontinuadas por dois motivos, um deles é por não ser possível o uso de uma chave estrangeira em uma super coluna, e a outra é pela performance, pois para a operação de leitura que qualquer coluna contida em uma super coluna era necessário carregar toda a super coluna para a memória [14].

Já na sua distribuição estável atual, versão 3.0, os elementos do modelo de dados são [1]: os *keyspaces*, as famílias de colunas, as tabelas, linhas e colunas, descritos mais detalhadamente a seguir:

Keyspace

O *Keyspace* é o agrupamento de dados que se pode ser comparado a um esquema em um banco de dados relacional, porém seu maior diferencial é de que o *keyspace* possui informações sobre como os dados serão replicados ao longo de cada computador no *cluster*, sendo eles o fator de replicação e a estratégia de replicação. O *keyspace* possui a maior granularidade dentre os modelos de dados, sendo que uma família de colunas deve estar contida dentro dele.

Famílias de colunas

As famílias de colunas são agrupamentos de colunas que podem ser comparados às

tabelas em um banco de dados relacional, porém o maior diferencial entre entre uma tabela do modelo relacional e da família de colunas do Cassandra é que, enquanto o modelo de uma tabela é fixo, na família de colunas é possível a adição de colunas à qualquer momento. Dentro de uma família estão contidas várias linhas. Dentro de uma família de colunas, a primeira coluna sempre é a chave primária, também conhecida como *row key*.

Linha

As linhas são um conjunto de colunas que possuem a mesma *row key*. Uma característica principal das linhas é que o mecanismo de armazenamento só aloca espaço para as colunas que possuem dados em uma linha, diferentemente de um banco relacional, no qual o espaço alocado para para uma coluna de uma linha que não possui dado deve ser escrita com um valor nulo. Dessa forma há um menor esforço computacional ao adicionar ou retirar uma coluna de uma tabela, além de viabilizar a prática de criação de colunas em tabelas já populadas.

Coluna

A coluna é o menor grão para armazenar dados, estando dentro de uma família de colunas e sendo composta pelos campos: nome, valor e um campo chamado *timestamp* que representa um valor de tempo para qual o dado foi escrito ou atualizado.

3.1.3 Operações do Banco

A escrita de dados no Cassandra é feita de forma singular e bem diferente do modelo relacional. Ao ser enviada para o banco uma operação de escrita, os dados são armazenados em uma estrutura, que fica em RAM, chamada *memtable*. Além disso, a própria instrução de escrita também é armazenada, só que em disco, em uma estrutura chamada *Commitlog*. Os *Commitlogs* servem principalmente para garantir que uma instrução seja realizada mesmo em caso de alguma falha. Durante o processo de escrita no banco, ao atingir um certo limite, configurável, a *memtable* despeja tudo que está na memória em disco para as *SSTables* [14]. Somente quando a operação de escrita na *SSTable* é executado com sucesso que o registro da instrução é apagada do *Commitlog*.

A Figura 3.4 demonstra graficamente o processo de escrita do Cassandra. Pode-se perceber que primeiramente o Cassandra manda a instrução de escrita, logo em seguida há o registro no *commitlog* juntamente com o carregamento dos dados na *memtable*, depois a confirmação de operação concluída para o usuário e por fim o despejo dos dados para a *SSTables*. O interessante a se perceber é que caso ocorra alguma falha no sistema antes do despejo para a *SSTables*, as operações já estão salvas no *commitlog*, então, ao se recuperar, o sistema reexecuta as instrução e as carregas na *memtable* novamente.

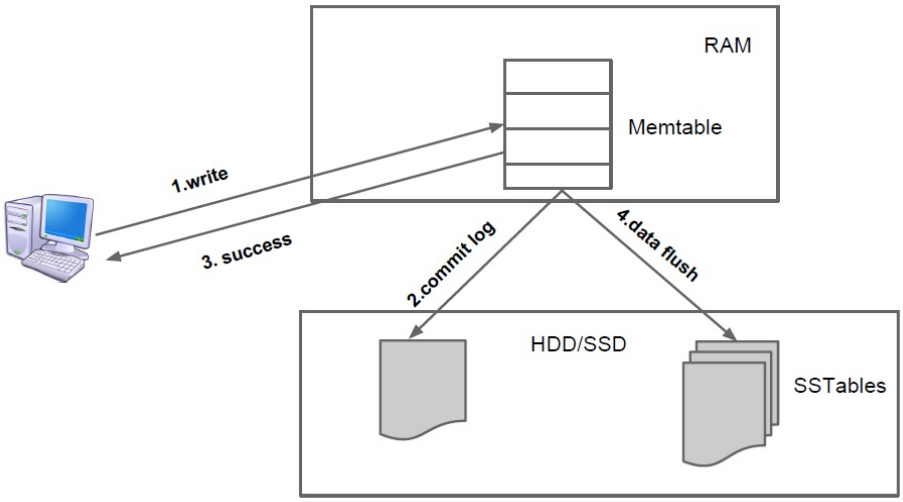

Figura 3.4: Processo de escrita do Cassandra. Adaptado de [14].

Ao executar uma instrução de leitura, é preciso agrupar tanto as *SSTables* quanto a *memtable* para retornar os registros correspondentes ao que foi pesquisado. Para auxiliar essa operação, uma estrutura chamada *Bloom Filter*, presente em cada *SSTable*, confere se a *SSTable* possui algum dado referente a consulta [14]. Outra estratégia que o Cassandra utiliza para melhorar a pesquisa é o uso de uma *cache* somente para as *row keys*. A *row cache*, como é chamada, carrega todas as *row keys* de determinada tabela para a memória quando buscada [14].

Pode-se dizer que o Cassandra não possui uma atual remoção das tabelas. Ao executar uma instrução de deletar a tabela é somente marcada na *SSTable* como uma apagada e um processo, chamado de compactação, que é executado concomitantemente com a instrução de deletar, agrupa fragmentos de linhas para otimizar a pesquisa de um dado [14]. O mesmo ocorre com o processo de atualização. O Cassandra marca o registro antigo como apagado e escreve o novo registro com um *timestamp* mais recente, portanto é possível acessar versões anteriores de dados já atualizado ou até mesmo apagados [14].

3.2 Arquitetura do Sistema

Para melhor entendimento de como é a arquitetura do Cassandra existem algumas estruturas chaves que devem ser descritas [14]:

- *Nó*: componente mais básico de uma estrutura Cassandra. Uma instância física do Cassandra. Não existe diferença entre dois nós, uma vez que o banco é descentralizado.

- *Data center*: um conjunto de nós que se relacionam entre si, tanto fisicamente quanto virtualmente. O *data center* é onde os parâmetros de replicação são configurados. É possível criar um *data center* de apenas um nó.
- *Cluster*: um conjunto de *data centers*, no qual o dado é armazenado. Pode existir um *cluster* com apenas um *data center*.
- *Replicação*: processo de armazenamento de cópias dos dados em vários nós para garantir as características de confiabilidade e tolerância a falhas. O número de cópias é definido pelo fator de replicação e o modo como os dados serão replicados pela estratégia de replicação.
- *Particionador*: serve para distribuir os dados de maneira uniforme entre os nós do *cluster*, para que a carga seja balanceada.
- *Cassandra Query Language (CQL)*: é uma linguagem de *script* usado para ser a interface do banco para o usuário. A abstração de uma tabela possuindo linhas e colunas é bastante semelhante com a da linguagem SQL usada em bancos relacionais. Apesar de várias semelhanças existem diversas diferenças entre o CQL e o SQL, uma diferença é que o CQL não suporta alguns recursos como *joins*, presentes em bancos que utilizam SQL.

O Cassandra é projetado para lidar com uma grande quantidade de dados em vários nós, buscando eliminar a possibilidade de erros. Sua arquitetura é fundada no entendimento de que falhas, tanto de sistema como de hardware, podem acontecer. A abordagem para tratar essas condições de falhas ocorrem através da utilização de um sistema distribuído em topologia de anel(como o *peer-to-peer*), no qual todos os nós estão na mesma posição hierárquica e os dados são distribuídos entre os nós do *cluster* [14].

A arquitetura do Cassandra permite que um usuário se conecte a qualquer nó em algum *data center* e acesse os dados utilizando a linguagem CQL. Normalmente, um *cluster* apresenta um *keyspace* por aplicação. Além das instruções CQL serem executadas através de um programa *shell*, provido pelo próprio Cassandra, chamado *cqlsh*, é possível utilizar *drivers* em diferentes linguagens de programação para que sejam feitas as execuções [14].

3.2.1 Distribuição e Replicação dos Dados

Como já dito anteriormente, o Cassandra possui a premissa de um sistema que está rodando em uma rede sem hierarquia entre nós. Esse sistema faz cópias dos dados e distribui as cópias entre um grupo de nós, portanto pode-se dizer que a distribuição e a replicação dos dados estão fortemente relacionados. Os dados são organizados por tabelas e identificados com uma chave primária. Essa chave primária determina em qual

nó os dados serão armazenados. As cópias das linhas também são escritas e chamadas de réplicas [14].

Existem alguns fatores que podem influenciar a replicação dos dados, sendo eles [14]: os nós virtuais, o particionador, a estratégia de replicação e o *snitch*. A estratégia de replicação está associada com o fator de replicação. Caso o fator de replicação seja dois, quer dizer que existem duas cópias de cada linha em nós distintos. A estratégia de replicação, por sua vez, define aonde essa outra linha será armazenada. Com a opção *SimpleStrategy*, a linha será armazenada no próximo nó a direita no anel no mesmo *datacenter*. Já a opção *NetworkTopologyStrategy* define que as replicas serão armazenadas em diferentes *data centers* [14].

O *snitch* define a qual *data center* e *rack* um nó pertence. O *Snitch* informa ao Cassandra sobre a topologia da rede para que uma leitura de dados seja feita da forma mais eficiente. Além disso o *Snitch* permite que o Cassandra distribua as réplicas em nós que são estejam no mesmo *rack*, afim de aumentar a disponibilidade dos dados [14].

Os outros dois fatores que influenciam a replicação dos dados, os nós virtuais e o particionador, são temas que merecem ser abordados com um pouco mais de detalhes e serão explicados a seguir[14]:

Nós virtuais

Nós virtuais, também conhecidos como *vnodes*, são representações virtuais de vários nós dentro um nó real. Antes da versão 1.2 do Cassandra, era necessário calcular e atribuir um *token*, um número inteiro que representava o início da memória disponível, para cada nó físico do *cluster*. Os *tokens* são definidos e distribuídos entre os nós pelo particionador de tal forma que os *tokens* são únicos, crescentes e que o valor do *token* subsequente determina o fim do espaço de memória do anterior e o início do espaço de memória em um próximo nó. Cada nó recebe apenas um *token*.

A partir do Cassandra 1.2, os *vnodes* ou nós virtuais mudaram essa abordagem. Eles permitem que um nó possa ter mais de um *token*. A quantidade de nós virtuais dentro de um mesmo nó físico dita quantos *tokens* um nó físico terá. Estes nós virtuais são uma maneira de simular um maior número de nós dentro de um nó real. Eles inclusive são tratados pelo particionador como um nó real e por isso trazem algumas vantagens:

- Os *tokens* são automaticamente calculados e distribuídos.
- No momento em que um nó é adicionado ou removido, o rebalanceamento do *cluster* é feito automaticamente. Quando um nó se junta ao *cluster*, ele assume a responsabilidade por uma parcela de dados de outros nós. Se um nó falhar, o seu conteúdo é distribuído uniformemente entre outros nós do *cluster*.

- É possível colocar um número proporcional de nós virtuais em máquinas com capacidades de armazenamento diferentes entre si, facilitando o uso de máquinas diferentes.
- A reconstrução de um nó que falhou é mais rápida, pois envolve todos os outros nós do *cluster*.

O Cassandra, como dito anteriormente, segue uma topologia de anel. A Figura 3.5 auxilia a demonstrar essa topologia, além de mostrar como é um *cluster* sem a utilização de nós virtuais. Cada nó na figura representa um computador, um nó físico. Sendo assim, o anel é formado pelos 6 nós físicos A, E, I, M, O e U.

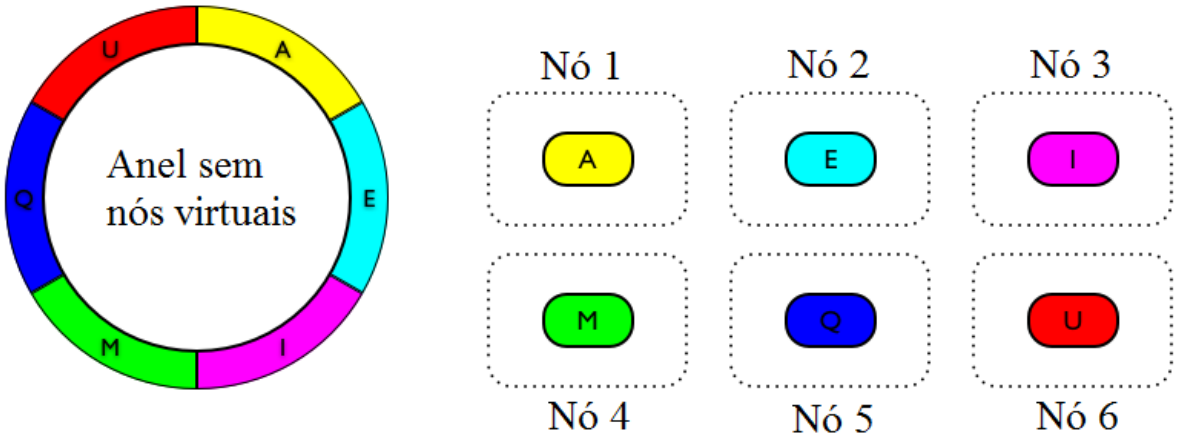

Figura 3.5: Exemplo de *Cluster* sem *vnodes*, adaptado de [14].

A Figura 3.6 demonstra o mesmo *cluster*, porém com a utilização dos nós virtuais. Cada nó físico se transforma em diversos nós virtuais dentro da topologia. Tomando como exemplo o Nó 1, temos que o nó físico A da Figura 3.5 foi subdividido em 4 nós virtuais, sendo eles A, X, W e V. O mesmo acontece em todos os outros nós. Isso faz com que o anel possua agora um total de 24 nós, utilizando apenas 6 computadores.

Particionador

A função primária do particionador é gerar e distribuir os *tokens* entre os nós, dessa forma ele determina como os dados são alocados nos nós do *cluster*. Cada nó possui um único e distinto *token*, pois é ele quem determina a extensão do armazenamento dos dados. Essa extensão que abrange um *token* a outro é uma abstração das chaves de uma *row key* atreladas a cada dado inserido no banco, que serão armazenadas naquele nó. Assim a extensão de um nó inicia-se no *token* do nó atual até o próximo valor de *token* no anel. O Cassandra possui três formas para gerar e distribuir esses *tokens*. Os principais particionadores são [14]:

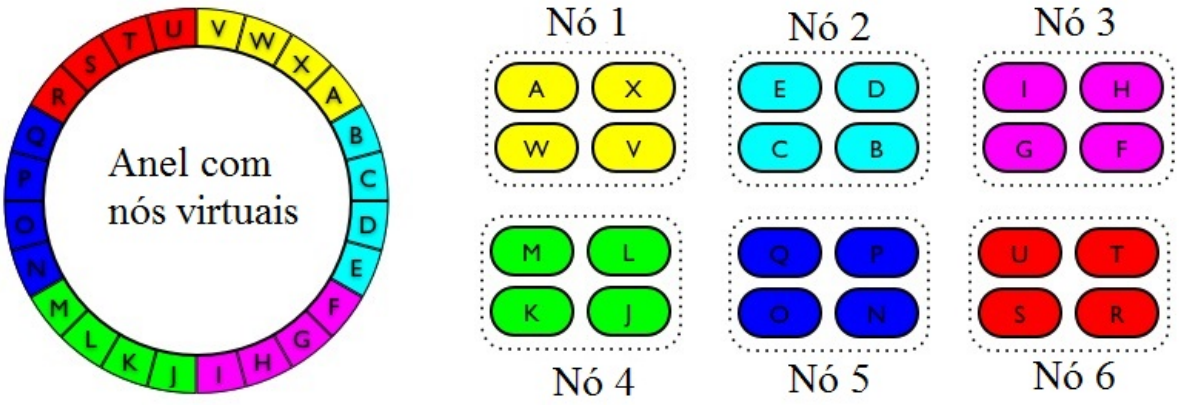

Figura 3.6: Exemplo de *Cluster* com *vNodes*, adaptado de [14].

- *RandomPartitioner*: era a seleção padrão até o Cassandra 1.2. Ele agia de forma a distribuir os dados de forma que estejam balanceados pelo anel. Essa distribuição é feita utilizando um *hash* MD5 sobre o valor de cada *row key*. Dessa forma os *tokens* gerados são do tipo inteiro com valores de 0 a $2^{127} - 1$.
- *Murmur3Partitioner*: a seleção padrão atual do Cassandra. Provê uma performance melhor que o *RandomPartitioner* e tão aleatório quanto, pois cria um *hash* de 64-bits da *row key*. Possui um intervalo de valores entre -2^{63} até $+2^{63} - 1$.
- *ByteOrderedPartitioner*: geralmente usado para partições ordenadas. Diferentemente das opção anteriores, as *row keys* não passam por uma função de *hash*, elas são ordenadas lexicalmente pelo primeiro *byte* da *row key*. Assim os *tokens* gerados são do tipo *string*. Uma particularidade desse método é que ele é passível de realizar pesquisas ordenadas. Por exemplo, caso uma *row key* possua nomes de usuários, é possível pesquisar linhas que estejam entre Marcella e Matheus. Essa *query* não seria possível com particionadores aleatórios, porque as chaves pesquisadas estariam armazenadas de acordo com seus *hashs*. Apesar dessa vantagem, esse particionador traz algumas dificuldades para outras tarefas do Cassandra. No caso das *row keys* serem muito similares, os valores dos *tokens* serão muito próximos e assim estes dados serão armazenados em um mesmo nó, fazendo com que o anel fique desbalanceado e criando um gargalo na consulta, pois não se estará utilizando o potencial das outras máquinas no anel.

A Figura 3.7 demonstra o uso de um particionador aleatório em um *cluster* usando

nós virtuais. Em um comparativo com a Figura 3.6, os dados na Figura 3.7 ficam mais dispersos e o anel melhor balanceado. Um exemplo para se explicar o funcionamento é que ao inserir 4 linhas, o Casandra inserirá cada uma delas em um *vnode* diferente, assim as 4 linhas serão distribuídas nos nós virtuais: A, B, C e D. Conforme a Figura 3.6 essas linhas estarão apenas nos nós 1 e 2, enquanto que no anel da Figura 3.7 essas mesmas linhas estão nos nós 3, 5, 2 e 6. A vantagem de se ter um anel bem distribuído é que a consulta é melhorada, uma vez que o número de consultas em paralelo é maior.

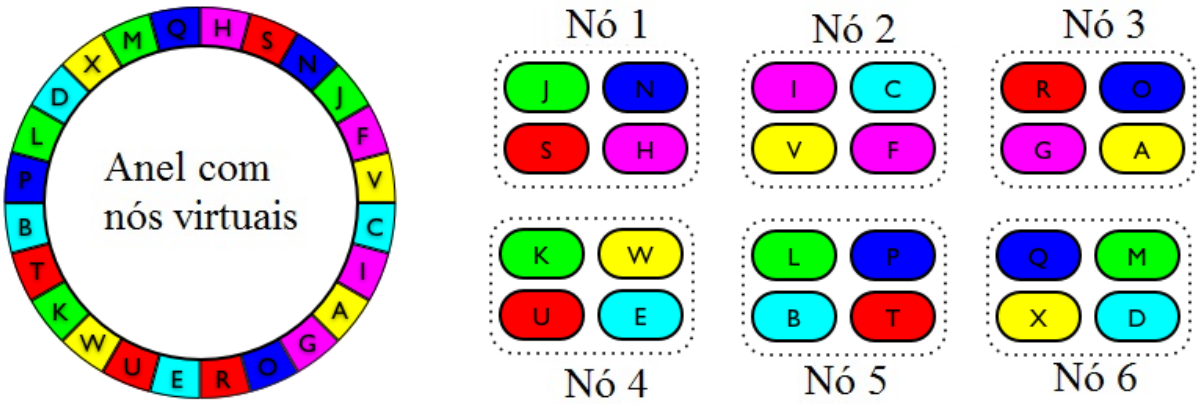

Figura 3.7: VNodes Particionados, adaptado de [14].

3.2.2 Níveis de Consistência

Consistência pode ser explicada como a forma na qual as linhas são mantidas atualizadas e sincronizadas nas cópias existentes. O objetivo é que sempre que for feita uma leitura de um dado no banco, este será lido e estará na sua versão mais recente. O Cassandra apresenta o conceito conhecido como *tunable consistency* em que para cada leitura ou escrita, seja possível escolher o quão consistentes os dados solicitados devem estar [19]. Isso é feito através de alterações no caminho dos dados e no número de cópias geradas. Além disso, o Cassandra também apresenta uma série de mecanismos embutidos para assegurar que os dados permaneçam corretos em suas diferentes réplicas [14]. Os níveis de consistência do Cassandra podem ser configurados de acordo com a necessidade da aplicação, sendo possível ajustar o *trade-off* entre disponibilidade em relação à precisão dos dados. A configuração da consistência pode ser em um *cluster*, *data center* ou até em operações individuais de leitura e escrita, sendo que o padrão definido pelo Cassandra é o nível *ONE*. Os níveis de consistência tanto para escrita quando para leitura são melhores descritos a seguir [14].

Níveis de Consistência de Escrita

Os níveis de consistência podem ser ajustados para conseguir o equilíbrio entre tempo de resposta e consistência. No caso da escrita, seguem alguns dos níveis e suas características:

- *ALL*: É o que possui a maior consistência e a menor disponibilidade entre todos os níveis. Porque quando uma escrita é executa, ela deve ser replicada em todos os nós do *cluster*, ou seja, ela é escrita em todos os *commit logs* e em todas as *memtables*.
- *QUORUM*: Fornece uma consistência forte entre os *clusters* de um *data center*, uma vez que uma instrução de escrita deve ser registrada no *commit log* e na *memtable* de um nó réplica em cada *data center*. Porém há possibilidade de falha.
- *ONE*: Uma escrita deve ser registrada no *commit log* e na *memtable* de apenas uma réplica. O nó de réplica responsável por atender o pedido é o nó mais próximo daquele que recebeu a *query*, a não ser que o sistema identifique que o nó possua um desempenho não aceitável, dessa maneira o controle é passado para outro nó. É utilizado pela maioria dos usuários, pois os requerimentos não são tão rigorosos de serem atendidos.
- *ANY*: Fornece baixa latência e a garantia de que uma escrita nunca irá falhar. Oferece a menor consistência e a maior disponibilidade em relação aos outros níveis. Um detalhe para esse nível é que caso os nós de réplica para uma dada escrita tenham falhado, o dado não poderá ser lido até que eles se recuperem.

Níveis de Consistência de Leitura

Os níveis de consistência para escrita são semelhantes aos utilizado pela leitura. Ao ler um dado, o Cassandra verifica o número de réplicas que enviaram os dados e quão recente eles são, essas medidas podem variar de acordo com os níveis descritos a seguir:

- *ALL*: Fornece a mais alta consistência de todos os níveis, porém possui a menor disponibilidade. Isso deve-se ao fato de que com esse nível o Cassandra só retorna uma resposta assim que todos as réplicas responderem. Caso uma réplica deixe de responder, a operação de leitura falha.
- *QUORUM*: Garante uma forte consistência dos dados através dos *data centers*. Uma vez que o dado é retornado assim uma réplica de cada *data center* tiver respondido. Possui um certo nível de falha, pois caso algum *data center* não responder, a operação falha.

- *ONE*: Fornece a maior disponibilidade entre os níveis, porém deve-se aceitar a possibilidade de leitura de dados antigos. A resposta é retornada pela réplica mais próxima, determinado pelo *snitch*.

Capítulo 4

Estudo de Caso e Análise de Resultados

Neste trabalho foi realizado um estudo de caso para avaliar o comportamento do sistema gerenciador de banco de dados Cassandra, quando utilizado para armazenar dados provenientes da execução de *workflows* científicos de Bioinformática. Para isso, foram executados três *workflows*, descritos na Seção 4.1.1, e implementado um programa necessário para otimizar as operações realizadas no banco de dados. A Seção 4.3 descreve os resultados obtidos pela inserção e extração dos dados das execuções. Por fim, a Seção 4.4 apresenta a análise dos dados mostrados na Seção 4.3.

4.1 Ambiente de Execução

As execuções dos *workflows* e dos testes de inserção e extração do banco de dados Cassandra foram todas realizadas em um mesmo ambiente. O laboratório utilizado disponibiliza de 15 máquinas, todas configuradas com as mesmas especificações. Cada máquina utiliza o sistema operacional Linux (Ubuntu 14.04 64bits), possui 8GB de memória RAM, 246.4 GB de disco rígido e um processador Intel® Core™ i5.

Foram montados três *clusters*, de 2, 4 e 6 computadores. Cada máquina foi configurada com um IP fixo dentro da rede da Universidade de Brasília. A Figura 4.1 demonstra o *cluster* de 6 máquinas, sendo que cada máquina é um nó Cassandra instalado e configurado.

4.1.1 *Workflows* de Bioinformática

Para avaliar o comportamento do banco de dados Cassandra, foram executados três *workflows* usados em projetos transcritômicos, a fim de obter formatos de arquivos diferentes,

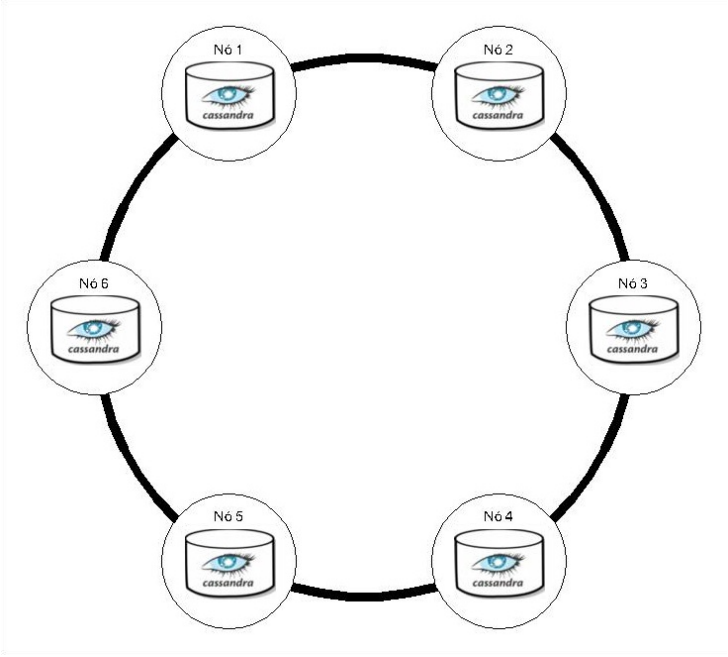

Figura 4.1: *Cluster* de 6 Máquinas.

assim como uma maior quantidade de dados para teste. Os *workflows* utilizados são descritos melhor a seguir.

O *workflow* 1 possui o objetivo de identificar genes diferencialmente expressos em células humanas cancerígenas do rim e do fígado [27]. Esse *workflow* é composto apenas pela fase de mapeamento e está ilustrado na Figura 4.2. Na figura cada programa é representado por uma caixa com seu respectivo nome. A esquerda dos programas estão os arquivos que serão utilizados como entrada e a direita os arquivos que resultaram após a execução do programa. A cor azul representa a fase de mapeamento.

O dado de entrada do *workflow* é um arquivo no formato FASTA, com tamanho de 115,5 MB, e foram obtidos arquivos no formatos SAM e BED como saída. Para executar o *workflow* foram utilizadas as ferramentas: Bowtie [4], sam2bed, genome2interval e coverageBed (*scripts* escritos na linguagem Perl) [27]. O tempo total de execução desde *workflow* foi 5 minutos e 31 segundos, gerando quatro arquivos, totalizando aproximadamente 36,7 MB.

O *workflow* 2 trata de dados de experimentos de RNA-Seq do fungos *Schizosaccharomyces pombe* [3], e é composto de três fases: montagem, mapeamento, anotação. Esse *workflow* pode ser demonstrado na Figura 4.3. As fases de montagem, mapeamento e anotação podem ser identificadas pelas cores azul, verde e laranja, respectivamente. Cada caixa da figura representa um programa utilizado e pode receber um ou mais arquivos como entrada e produzir um ou mais arquivos como saída. As ferramentas utilizadas em

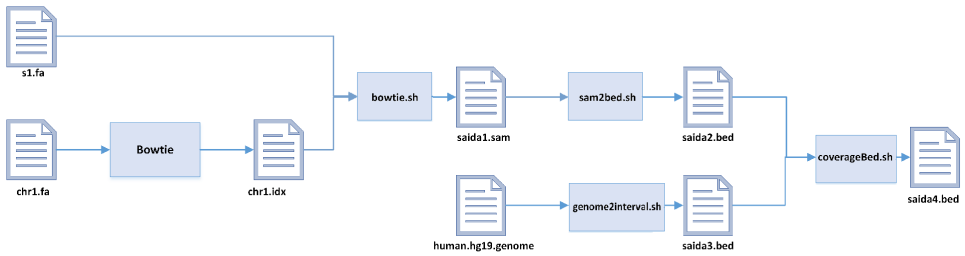

Figura 4.2: *Workflow* 1.

cada fase foram Trinity [39], Bowtie, TopHat [9] e Blast [8].

Foram utilizados oito arquivos de entrada, todos no formato FASTQ, com tamanho total de 1,35 GB, os quais foram divididos em dois grupos: *reads* de direita (quatro arquivos concatenados em um) e *reads* de esquerda (igual ao outro grupo), os arquivos de entrada possuem um tamanho de 1,35 GB.

Os arquivos obtidos em cada fase do *workflow* 2 totalizaram aproximadamente 1,04 GB. Os arquivos possuem formatos FASTA, FASTQ, BAM, BED e TXT, e o tempo total gasto para a execução completa foi 16 horas 50 minutos e 51 segundos.

O terceiro *workflow* também trata de dados de experimentos de RNA-Seq, porém utiliza o fungo *Aspergillus fumigatus* [34]. A Figura 4.4 é um ilustrativo de tal *workflow* que possui as fases de filtragem, mapeamento, montagem e anotação, demonstradas pelas cores vermelho, azul, verde e laranja, respectivamente. Foram utilizados 6 arquivos no formato FASTQ com tamanho total de 17,33 GB que ao passar pela filtragem geram 12 arquivos (6 FASTA e 6 FASTQ), os quais são divididos em quatro blocos. Os dois primeiros blocos, formados pelos arquivos do tipo FASTA concatenam-se e são utilizados para a fase de mapeamento, enquanto os dois últimos, do tipo FASTQ, para a fase de montagem. Os resultados da montagem vão para a fase de anotação, na qual são gerados dois arquivos de texto. A execução total do *workflow* gastou um total de 23 horas 28 minutos e 48 segundos. Os arquivos de saída somaram aproximadamente 13,89 GB, sendo que seus formatos foram iguais aos do *workflow* 2 e as ferramentas utilizadas foram Prinseq-Lite [51], Bowtie, TopHat, Trinity e Blast.

Após a execução dos *workflows*, foram selecionados quais arquivos deveriam ser salvos nos bancos de dados. A escolha desses arquivos foi tomada baseada em uma pesquisa feitas com biólogos da área de Bioinformática. Os arquivos escolhidos permitem a reexecução do *workflow* a partir de qualquer fase, assim como também fazer análises pertinentes aos *workflows*. A Tabela 4.1 sintetiza as informações de cada *workflow*, mostrando o tempo gasto para execução do *workflow* (Tempo), expresso em horas, minutos e segundos (hh:mm:ss), o tamanho dos arquivos de entrada (Tam. Entrada), o tamanho dos arquivos

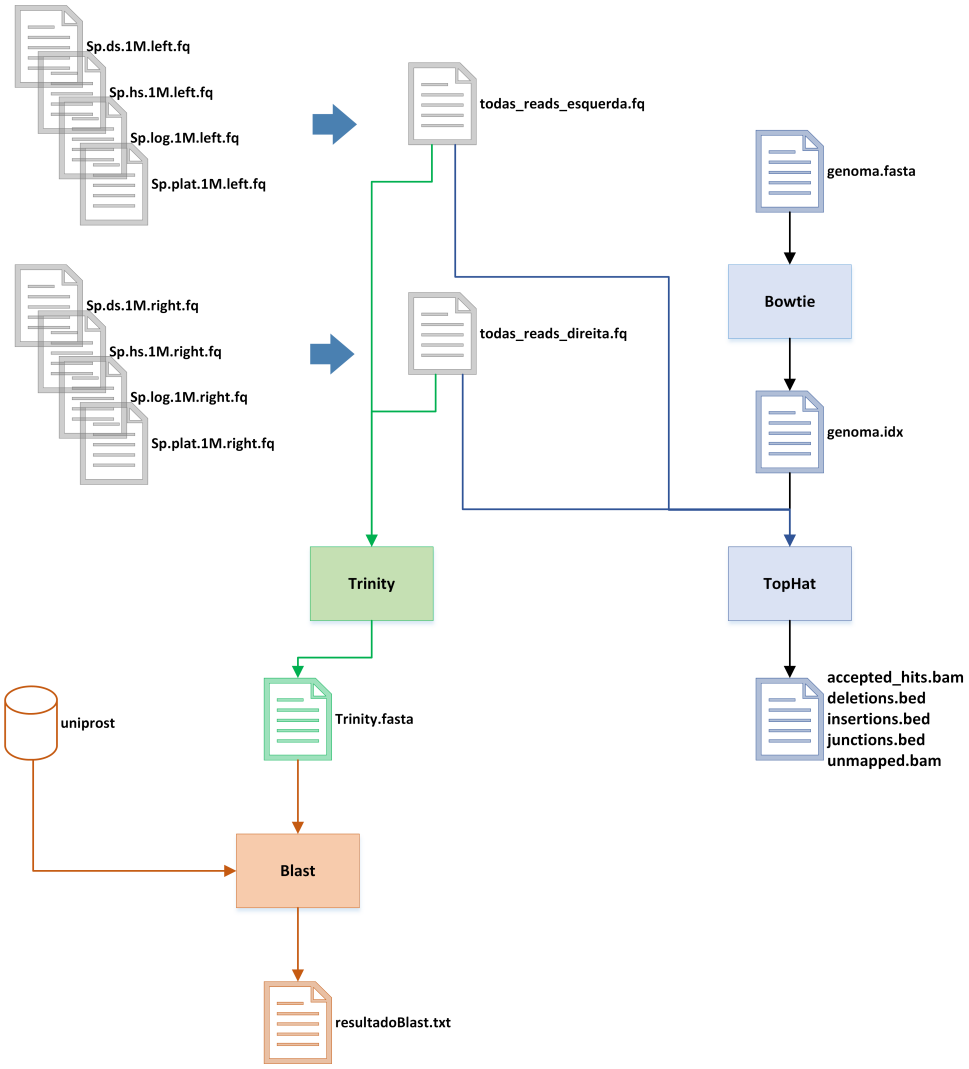

Figura 4.3: *Workflow* 2.

de saída (Tam. Saída) e o tamanho total (Tam. Total).

4.2 Aplicação Cliente

Concomitantemente a execução dos *workflows*, desenvolveu-se uma aplicação para realizar a importação e a exportação dos dados do Cassandra. Esta aplicação foi desenvolvida na linguagem Java, utilizando como ambiente de desenvolvimento o Eclipse juntamente com o Cassandra Java Driver [16], uma biblioteca disponibilizada pela Datastax. A aplicação desenvolvida tem os seguintes requisitos funcionais:

- criar um *keyspace* para a aplicação;

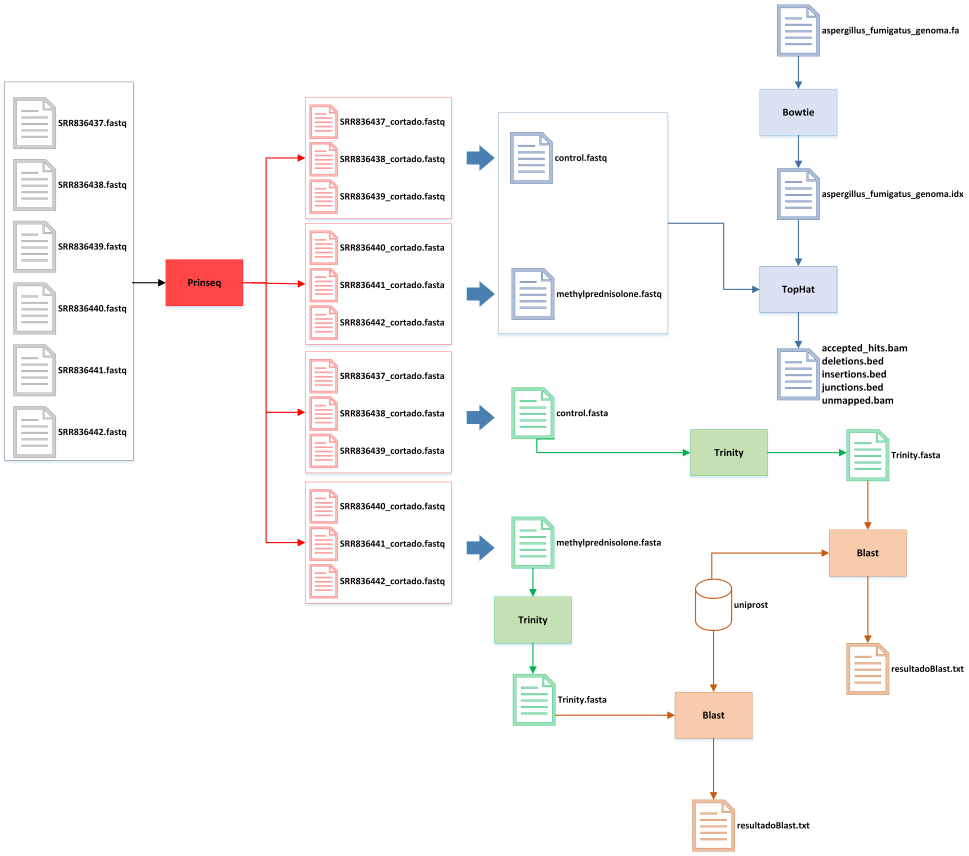

Figura 4.4: *Workflow* 3.

Tabela 4.1: Arquivos Armazenados nos Bancos de Dados

Workflow	Tempo	Tam. Entrada	Tam. Saída	Tam. Total
Workflow 1	00:05:31	115,5 MB	36,7 MB	152,2 MB
Workflow 2	00:50:51	1,35 GB	1,04 GB	2,39 GB
Workflow 3	23:28:48	17,33 GB	13,89 GB	31,22 GB

- criar uma tabela para armazenar as informações básicas (metadados) de cada arquivo;
- criar uma tabela para armazenar os arquivos propriamente ditos;
- receber um arquivo e o armazenar juntamente com seus metadados, cada qual na sua respectiva tabela;
- extrair um arquivo especificado do banco de dados;
- comparar o arquivo original com o arquivo extraído.

A aplicação desenvolvida inicialmente tenta se conectar a uma instância do Cassandra. Após isso, ela cria um *keyspace* chamado *bioinformaticKeySpace* e duas tabelas, sendo uma

para armazenar o arquivo que será importado, chamada de *files*, e a outra que armazenará os metadados daquele arquivo, chamada de *meta*. Os metadados definidos foram o nome e a fase do *workflow*, o nome do arquivo, a data que ele foi inserido, o programa utilizado, a versão do programa e o tamanho do arquivo.

A estratégia de alocação das réplicas determina se elas serão distribuídas por uma rede de diferentes *cluster*, no caso a "Estratégia de Topologia em Rede", ou se serão distribuídas em um único *cluster*, no caso da "Estratégia Simples". Foi escolhida a Estratégia Simples por ser mais adequada ao ambiente de testes utilizado, uma vez que para cada execução dos testes, um único *cluster* foi utilizado. A Figura 4.5 mostra o modelo proposto, no qual dentro do *cluster* Cassandra está o *keyspace* criado. Nesse *keyspace* estão as duas tabelas utilizadas pelo programa, a tabela *meta* e a tabela *files*. Na tabela *meta* cada linha representa um arquivo que possui os dados associados aos seus respectivos metadados, descritos anteriormente. Já na tabela *files*, a *row key* 1 identifica o arquivo e as colunas seriam todos os pedaços, em binário, dos arquivos. Cada arquivo possui um tamanho diferente, logo o número de colunas varia de acordo com o tamanho do arquivo.

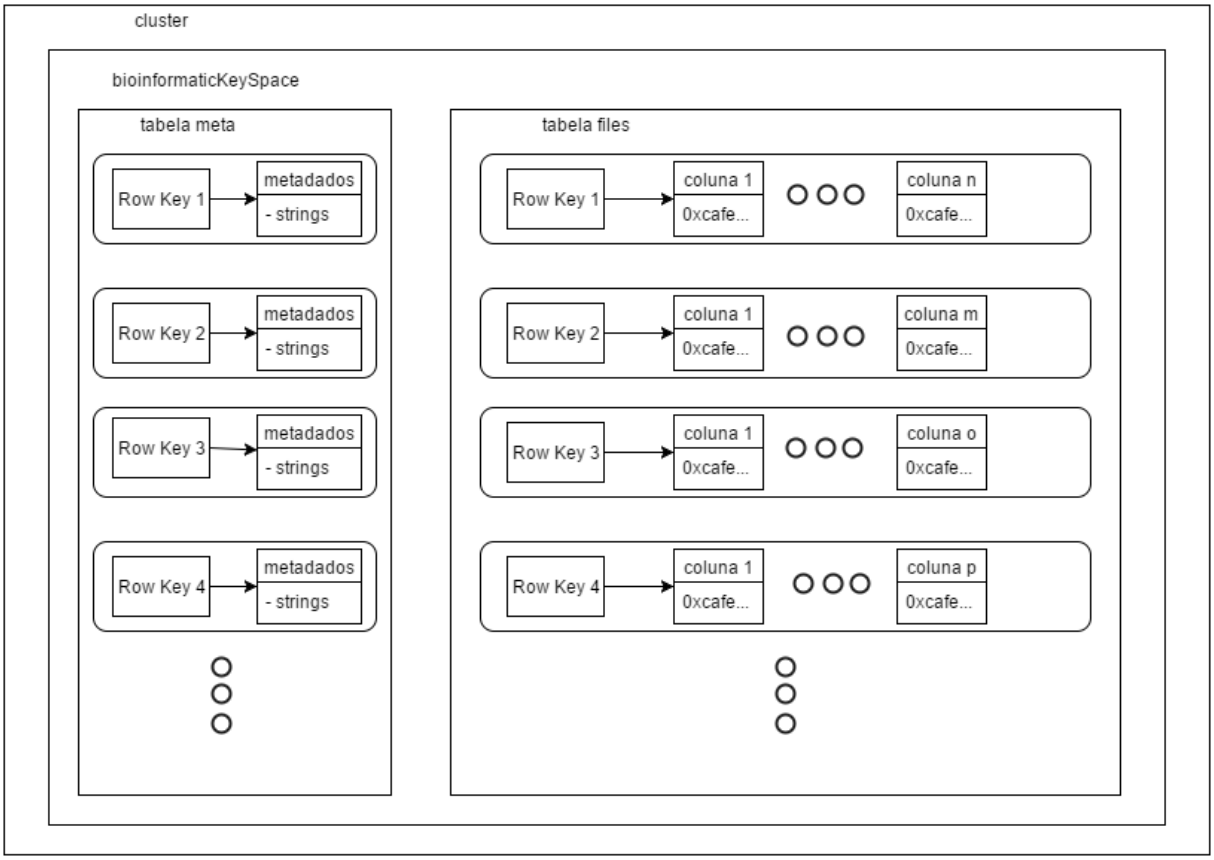

Figura 4.5: Exemplo do Modelo de Dados Proposto.

Depois de criado todo o esquema do banco de dados, é feita a inserção do arquivo. A metodologia por trás da inserção, que pode ser demonstrado através da Figura 4.6, é a seguinte: abre-se o arquivo em modo binário e pedaços desse arquivos são carregados em memória. Esses são colocados em um *buffer* de tamanho fixo. Em seguida é montada uma *query* de inserção com os metadados do arquivo, o *buffer* preenchido e um *timestamp* que representa a hora atual do sistema, dessa forma a primeira parte do arquivo é gravada. Em seguida o mesmo processo deve ser realizado com as outras partes que estão em memória, repetindo o processo até o fim arquivo.

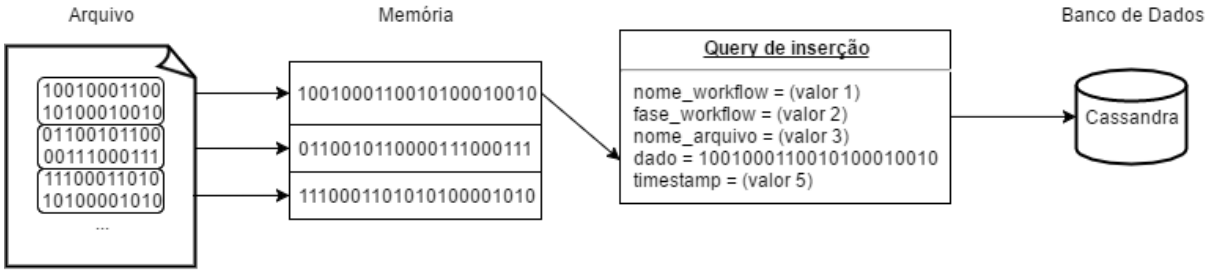

Figura 4.6: Exemplo de Inserção.

Ao final da inserção do arquivo no banco de dados, a aplicação registra esse arquivo na tabela *meta*, a fim de facilitar pesquisas, como por exemplo, saber quantos arquivos existem registrados no banco ou quais são os arquivos que pertencem a determinado *workflow*.

A extração é feita através da chave de cada arquivo ordenada pelo *timestamp*, ou seja, é retornado apenas uma linha com toda a sua família de colunas, sendo que as colunas na família de colunas estão ordenadas de acordo com o *timestamp* feito durante a inserção. A partir de então recupera-se cada coluna que possui o dado binário e o escreve em um arquivo de saída. Para garantir a integridade dos dados após a extração utilizou-se a ferramenta *diff*, nativa do sistema operacional, que compara dois arquivos linha a linha e aponta se existe alguma diferença.

Para uma análise do tempo de processamento de inserção e extração de dados da Bioinformática no Cassandra foram realizados diferentes testes variando os fatores de replicação. Desta forma, a aplicação foi executada em 8 ambientes diferentes, sendo eles:

- Fator de replicação 1 em três *clusters* (2, 4 e 6 máquinas);
- Fator de replicação 2 em três *clusters* (2, 4 e 6 máquinas);
- Fator de replicação 3 em dois *clusters* (4 e 6 máquinas). O *cluster* com 2 máquinas foi desconsiderado devido a impossibilidade de existirem 3 réplicas do dado com apenas 2 máquinas.

Tabela 4.2: Tamanho dos Arquivos do *Workflow*.

Workflow	Arquivos	Tamanho dos Arquivos
Workflow 1	Dados Brutos	115,5 MB
	Mapeamento	36,7 MB
Workflow 2	Dados Brutos	1,35 GB
	Montagem	11 MB
	Mapeamento	422,32 MB
	Anotação	610 MB
Workflow 3	Dados Brutos	17,33 GB
	Filtragem	11,62 GB
	Montagem	21,7 MB
	Mapeamento	1,25 GB
	Anotação	1 GB

Os teste das inserções e extrações dos arquivos para cada ambiente foram executados de duas formas diferentes, chamadas de execuções frias e execuções quentes. As execuções frias foram realizadas logo após todas as máquinas serem ligadas e o *cluster* integrado pela primeira vez, sem que houvesse nenhuma interação com o banco de dados anteriormente. As execuções quentes foram realizadas quando o banco de dados já estava sendo utilizado. Foram realizadas cinco execuções frias e cinco execuções quentes para cada *workflow* em cada um dos ambientes. Além disso, a fim de evitar distorções, foi desconsiderado o maior e o menor tempo para o cálculo da média.

4.3 Resultados Obtidos

Inicialmente, com objetivo de testar a aplicação foram realizados testes de criação de *keyspaces*, tabelas, inserção e retirada de dados a nível de *localhost*. Esses teste trouxeram resultados positivos, mostrando-se possível a execução dessas operações a nível de *cluster*.

A Tabela 4.2 mostra o tamanho dos arquivos dos *workflows* utilizados para realizar as inserções no banco de dados separados pelos arquivos de entrada iniciais do *workflow*, que são chamados de arquivo bruto, e pelos arquivos de saída de suas fases. Como pode ser observado no *workflow* 1 tem-se o arquivo bruto de 115,5 MB e os dados de saída da fase de mapeamento possuem 36,7 MB.

4.3.1 Inserções

Primeiramente, foram realizadas as inserções dos arquivos dos *workflows*, iniciando-se pelos dados brutos, e em seguida pelos arquivos de saída de cada uma das suas fases.

Tabela 4.3: Inserção dos Dados Brutos.

		Inserção dos dados brutos							
		2 Máquinas		4 Máquinas			6 Máquinas		
Teste	W	FR1	FR2	FR1	FR2	FR3	FR1	FR2	FR3
Q	W1	00:06	00:19	00:09	00:11	00:25	00:18	00:22	00:26
	W2	01:23	01:51	02:14	03:49	04:02	03:07	04:22	04:30
	W3	06:54	08:02	12:37	18:17	19:49	24:28	24:41	24:49
F	W1	00:07	00:16	00:07	00:07	00:24	00:14	00:22	00:25
	W2	01:24	01:54	03:35	03:45	03:57	04:08	04:10	04:25
	W3	07:30	07:46	12:54	18:37	19:30	24:11	24:36	24:58

Os dados brutos são essenciais para a reexecução de um *workflow*, sendo os arquivos iniciais de um *workflow*, e utilizados pela sua primeira fase. A Tabela 4.3 mostra a média dos tempos, expresso em minutos e segundos (mm:ss), obtidos nos testes quentes (Q) e frios (F) de cada *workflow* (W) nos *clusters* com duas, quatro e seis máquinas em seus respectivos fatores de replicação (FR). Pode-se observar pela tabela que o tempo médio para inserção dos dados brutos do *workflow* 1 em um *cluster* de 2 máquinas com fator de replicação 1, durante um teste quente, foi de 6 segundos, já a inserção para o mesmo *workflow*, no mesmo *cluster* com fator de replicação 2, durante o teste quente, foi de 19 segundos. Nessa mesma análise, para os teste frios, pode-se observar os valores de 7 segundos para o fator de replicação 1, e 16 segundos para o fator de replicação 2. Também é possível notar que os tempos entre as inserções dos testes quentes e frios são semelhantes e esta característica será melhor explorada durante a Seção 4.4.

Em cada fase são gerados arquivos com diferentes formatos, e por isso é importante fazer uma análise para as saídas de cada fase para mensurar qual o impacto do tipo do arquivo na inserção. A Tabela 4.4 contém os dados obtidos pela inserção dos arquivos de saída de cada fase do *workflow*. Essa tabela mostra a média dos tempos, expresso em minutos e segundos (mm:ss), obtidos nos testes quentes (Q) e frios (F) de cada *workflow* (W) nos *clusters* com duas, quatro e seis máquinas, de acordo com as fases dos *workflow* (FW), sendo divida em Filtragem (F), Montagem (Mo), Mapeamento (Ma) e Anotação (A), em seus respectivos fatores de replicação (FR). Pela tabela, pode-se observar que o tempo médio de inserção dos arquivos de saída da fase do mapeamento do *workfow 1*, durante um teste quente, em um *cluster* de 2 máquinas com fator de replicação 1 foi de 4 segundos, já para o o *cluster* de 2 máquinas com fator de replicação 2 foi de 6 segundos. Como é possível ver no *workflow* 3, a fase de filtragem foi a mais demorada, levando 14 minutos e 52 segundos, no fator de replicação em um *cluster* de duas máquinas. Isso era esperado, uma vez que essa fase possui os arquivos de maiores tamanhos.

A Tabela 4.5 refere-se ao somatório dos tempos, expresso em horas, minutos e segundos (h:mm:ss), para inserir os dados brutos com os arquivos das fases (Tabela 4.3 somada a

Tabela 4.4: Inserção dos Arquivos de Saída de Cada Fase.

			Inserção dos arquivos de saída das fases							
			2 Máquinas		4 Máquinas			6 Máquinas		
Teste	W	FW	FR1	FR2	FR1	FR2	FR3	FR1	FR2	FR3
Q	W1	Ma	00:04	00:06	00:04	00:05	00:06	00:04	00:06	00:08
	W2	Mo	00:01	00:01	00:01	00:01	00:02	00:02	00:02	00:02
		Ma	00:33	00:44	01:02	01:06	01:19	01:26	01:43	02:00
		A	00:43	00:52	01:14	01:33	01:37	01:41	01:54	01:55
	W3	F	14:52	15:00	19:01	29:41	31:03	36:40	37:20	37:11
		Mo	00:02	00:02	00:03	00:02	00:03	00:04	00:04	00:04
		Ma	01:39	02:21	02:42	03:42	05:00	04:56	05:06	05:56
		A	01:20	02:02	01:55	02:11	02:37	03:05	03:10	03:17
F	W1	Ma	00:04	00:05	00:04	00:04	00:06	00:04	00:06	00:07
	W2	Mo	00:01	00:01	00:01	00:01	00:02	00:02	00:02	00:02
		Ma	00:35	00:42	00:53	00:55	01:18	01:18	00:40	01:52
		A	00:42	00:45	00:58	01:25	01:39	01:32	01:51	01:52
	W3	F	13:13	13:35	18:28	27:20	29:18	36:38	37:10	36:38
		Mo	00:02	00:02	00:02	00:03	00:03	00:04	00:04	00:04
		Ma	01:48	01:52	02:22	04:01	04:51	04:51	05:04	05:55
		A	01:19	01:18	01:38	02:06	02:18	03:06	03:06	03:19

Tabela 4.4) de cada *workflow* (W) em cada *cluster*, sendo FR1, FR2, FR3 os fatores de replicação 1, 2 e 3, respectivamente. Pode-se analisar nessa tabela que o tempo médio total para a inserção dos arquivos do *workflow* 1, em um teste quente, num *cluster* de 2 máquinas com fator de replicação 1 é de 10 segundos, enquanto para o fator de replicação 2 é de 24 segundos. Também é possível observar que quanto maior o fator de replicação, maior é o tempo de inserção.

Tabela 4.5: Tempos de Inserção dos Dados Brutos Somados aos da Fase.

		Inserção							
		2 Máquinas		4 Máquinas			6 Máquinas		
Teste	W	FR1	FR2	FR1	FR2	FR3	FR1	FR2	FR3
Q	W1	0:00:10	0:00:24	0:00:14	0:00:16	0:00:31	0:00:23	0:00:28	0:00:34
	W2	0:02:40	0:03:28	0:04:31	0:06:30	0:07:00	0:06:15	0:08:01	0:08:26
	W3	0:24:47	0:27:28	0:36:18	0:53:53	0:58:33	1:09:13	1:10:21	1:11:16
F	W1	0:00:11	0:00:21	0:00:11	0:00:11	0:00:30	0:00:18	0:00:28	0:00:32
	W2	0:02:42	0:03:22	0:05:27	0:06:06	0:06:56	0:07:00	0:06:43	0:08:11
	W3	0:23:52	0:24:34	0:35:24	0:52:07	0:56:00	1:08:49	1:10:01	1:10:54

Tabela 4.6: Extração dos Dados Brutos.

		Extração dos dados brutos							
		2 Máquinas		4 Máquinas			6 Máquinas		
Teste	W	FR1	FR2	FR1	FR2	FR3	FR1	FR2	FR3
Q	W1	00:07	00:06	00:06	00:05	00:05	00:03	00:03	00:01
	W2	01:34	01:09	01:04	00:48	00:43	00:47	00:34	00:28
	W3	06:52	06:33	06:57	06:17	04:58	06:08	05:34	04:31
F	W1	00:07	00:06	00:07	00:04	00:04	00:02	00:03	00:02
	W2	01:36	01:07	00:57	00:49	00:49	00:47	00:32	00:34
	W3	06:44	06:27	06:59	06:09	05:08	05:58	05:17	04:32

4.3.2 Extração

Em seguida foram realizadas as extrações dos arquivos dos *workflows*. A Tabela 4.6 contém os dados obtidos somente pela extração dos dados brutos, enquanto a Tabela 4.7 contém os dados obtidos pela extração dos arquivos de cada fase do *workflow*.

A Tabela 4.6 mostra a média dos tempos, expresso em minutos e segundos (mm:ss), obtidos nos testes quentes (Q) e frios (F) de cada *workflow* (W) nos *clusters* com duas, quatro e seis máquinas em seus respectivos fatores de replicação (FR) para a extração dos dados brutos. Ao analisar a tabela nota-se que o tempo médio de extração dos dados brutos do *workflow* 1 em um *cluster* de máquinas com fator de replicação 1, durante um teste quente, foi de 7 segundos, enquanto para o fator de replicação 2 foi de 6 segundos, sendo igual aos resultados obtidos pelos testes frios.

A Tabela 4.4 mostra a média dos tempos de extração, expresso em minutos e segundos (mm:ss), obtidos nos testes quentes (Q) e frios (F) de cada *workflow* (W) nos *clusters* com duas, quatro e seis máquinas, de acordo com as fases dos *workflow* (FW), sendo divida em Filtragem (F), Montagem (Mo), Mapeamento (Ma) e Anotação (A), em seus respectivos fatores de replicação (FR). Pela tabela, o tempo médio de extração dos arquivos da fase de mapeamento do *workflow* 1 em um *cluster* de 2 máquinas com fator de replicação 1, em um teste quente, foi de 4 segundos. Enquanto para o fator de replicação 2, foi, também, de 4 segundos.

A Tabela 4.8 refere-se ao somatório dos tempos, expresso minutos e segundos (mm:ss), para extrair os arquivos de entrada com os arquivos das fases (Tabela 4.6 somada a Tabela 4.7) de cada *workflow* (W) em cada *cluster*, sendo FR1, FR2, FR3 os fatores de replicação 1, 2 e 3, respectivamente. Pode-se analisar nessa tabela que o tempo médio total de extração de todos os arquivos do *workflow* 1, em um teste quente, num *cluster* de 2 máquinas com fator de replicação 1 é de 11 segundos, enquanto para o fator de replicação 2 é de 10 segundos.

Tabela 4.7: Extração dos Arquivos de Cada Fase.

			Extração da saída das fases							
			2 Máquinas		4 Máquinas			6 Máquinas		
Teste	W	FW	FR1	FR2	FR1	FR2	FR3	FR1	FR2	FR3
Q	W1	Ma	00:04	00:04	00:04	00:04	00:04	00:04	00:04	00:04
	W2	Mo	00:01	00:01	00:01	00:01	00:01	00:01	00:01	00:01
		Ma	00:48	00:33	00:50	00:38	00:22	00:27	00:18	00:11
		A	00:31	00:23	00:32	00:30	00:15	00:24	00:20	00:11
	W3	F	10:33	09:54	10:35	08:10	05:03	10:06	07:59	04:12
		Mo	00:02	00:02	00:02	00:02	00:02	00:02	00:02	00:02
		Ma	02:39	02:26	02:30	01:52	01:45	02:09	01:31	00:40
		A	00:51	00:47	00:51	00:39	00:21	00:39	00:32	00:19
F	W1	Ma	00:04	00:04	00:04	00:04	00:04	00:04	00:04	00:04
	W2	Mo	00:01	00:01	00:01	00:01	00:01	00:01	00:01	00:01
		Ma	00:47	00:33	00:43	00:30	00:21	00:33	00:17	00:12
		A	00:30	00:23	00:30	00:26	00:17	00:26	00:22	00:12
	W3	F	10:39	09:52	10:42	09:46	04:41	10:02	07:42	04:32
		Mo	00:02	00:02	00:02	00:02	00:02	00:02	00:02	00:02
		Ma	02:37	02:19	02:41	02:09	01:41	02:23	01:25	00:42
		A	00:49	00:45	00:50	00:44	00:20	00:46	00:31	00:20

Tabela 4.8: Tempos de Extração dos Arquivos de Entrada Somados aos da Fase.

		Extração							
		2 Máquinas		4 Máquinas			6 Máquinas		
Teste	W	FR1	FR2	FR1	FR2	FR3	FR1	FR2	FR3
Q	W1	00:11	00:10	00:10	00:09	00:09	00:07	00:07	00:05
	W2	02:54	02:06	02:27	01:56	01:21	01:39	01:13	00:51
	W3	20:57	19:42	20:56	17:00	12:08	19:03	15:38	09:44
F	W1	00:11	00:10	00:11	00:08	00:08	00:06	00:07	00:06
	W2	02:54	02:03	02:11	01:47	01:28	01:47	01:12	00:58
	W3	20:51	19:26	21:14	18:50	11:52	19:11	14:56	10:08

4.4 Análise dos Dados

Como dito anteriormente foram realizadas cinco execuções para cada *workflow* em cada ambiente. Esses cinco tempos obtidos pelas execuções geraram algumas variações, pois o laboratório utilizado não era exclusivo para o estudo contido nessa monografia. Dessa forma, a média realizada desconsiderou o maior e o menor tempo. O gráfico da Figura 4.7 mostra os cinco tempos de inserção e de extração, expresso em horas, minutos e segundos (hh:mm:ss), obtidos na execução da filtragem do *workflow* 3 durante o teste quente, em um *cluster* de 4 máquinas e fatores de replicação 2. Como pode ser visto pelo gráfico, o maior valor para inserção é de 30 minutos e 8 segundos, enquanto o menor é de 28 minutos e 38 segundos. Já para a extração o maior valor é de 11 minutos e o menor de 7 minutos e 15 segundos. Esses quatro valores foram desconsiderados e foi realizado a média dos três tempos restantes da inserção e extração, resultando em uma média de 29 minutos e 41 segundos para inserção, como pode ser observado na Tabela 4.4, e de 8 minutos e 10 segundos para extração, como pode ser observado na Tabela 4.7.

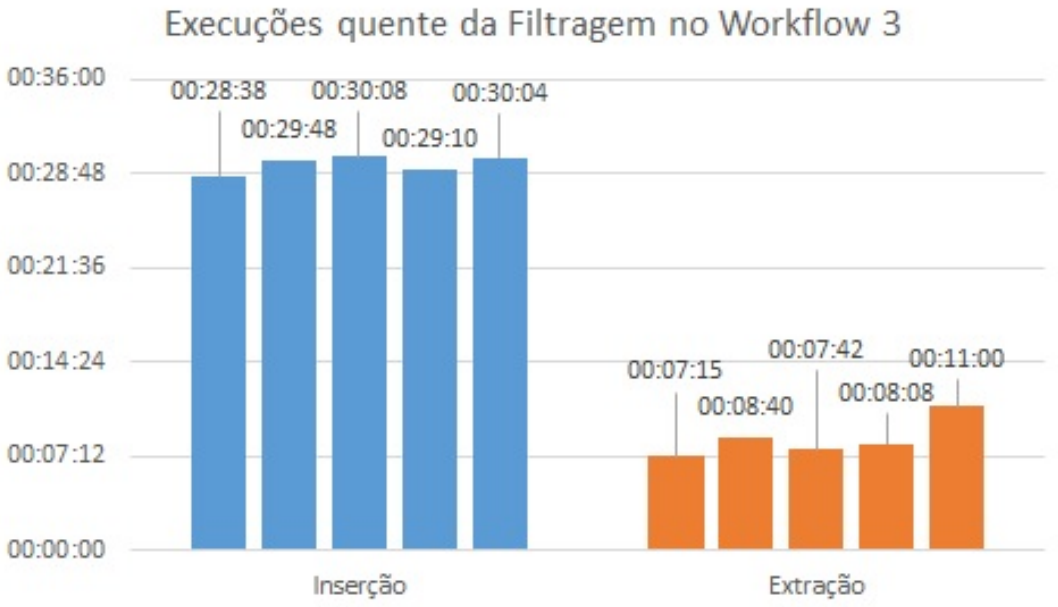

Figura 4.7: Execuções da Filtragem no *Workflow* 3.

Os resultados obtidos pelas execuções quente e fria, tanto para inserção quanto para extração, não variaram muito. O gráfico, gerado a partir das Tabelas 4.5 e 4.8, na Figura 4.8 apresenta uma análise feita dos testes quente e frio do *workflow* 3 para a inserção. Como pode ser visto no gráfico não há uma grande variação entre as duas barras, porém as barras da execução quente são sempre maiores que a da execução fria. Isso se dá ao

fato de que durante as execuções quente a *memtable* do Cassandra está cheia e necessita realizar o *dump* para a *SSTable* um maior número de vezes do que nas execuções frias.

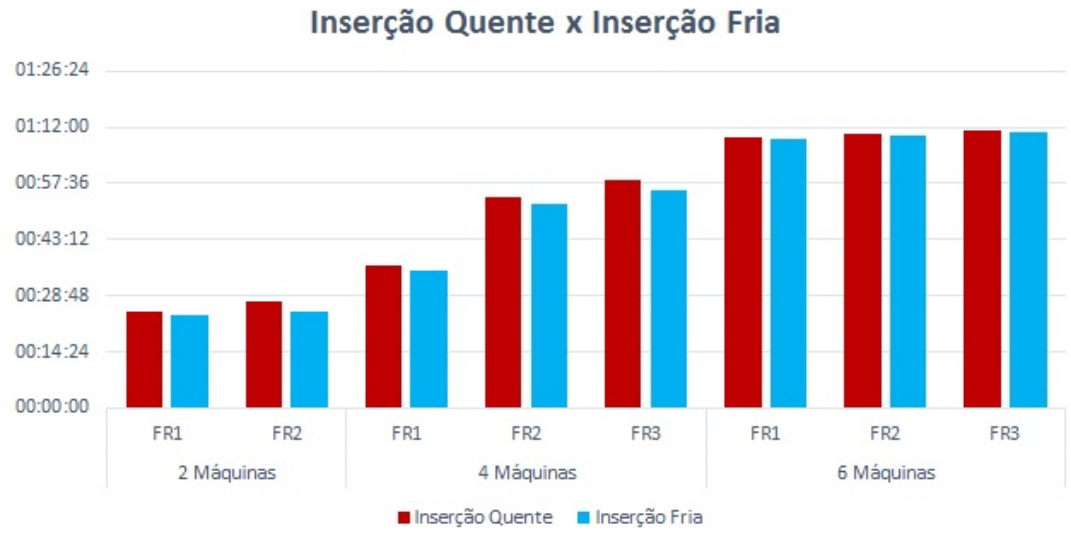

Figura 4.8: Inserção Quente x Inserção Fria.

Pela Tabela 4.4 pode-se perceber que na fase de montagem dos *workflows* 2 e 3 os tempos de inserção arquivos dos arquivos é muito próximo de zero, independente do número de máquinas utilizadas ou do fator de replicação. Essa mesma característica ocorre também com a fase de mapeamento do *workflow* 1. A Tabela 4.2 mostra que os arquivos de tais fases são arquivos que possuem tamanho menores que 120 MB, isso significa que para armazenar arquivos com essa faixa de tamanho pode-se aumentar o número de replicação e máquinas utilizadas, de forma a garantir uma confiabilidade dos dados maior, sem perder performance.

De acordo com a Tabela 4.5 e com o auxílio do gráfico da Figura 4.9, nota-se que quanto maior o número de máquinas e o fator de replicação, maior o tempo de inserção. A figura mostra os tempos de inserção do *workflow* 3 em todos os ambientes analisados, pode-se observar, por exemplo, que no *cluster* de 4 máquinas houve um aumento considerável do fator de replicação 1 para o fator de replicação 2, de 36 minutos e 18 segundos para 53 minutos e 53 segundos, respectivamente. Isso deve-se ao fato de que a resposta de finalização da inserção, ocorre quando o nó que está inserindo o dado recebe uma resposta indicando a conclusão da inserção de todas as réplicas. Outro fato para o aumento do tempo de inserção são as condições da rede, pois os dados devem trafegar pela rede para serem inseridos nas máquinas usadas como réplicas.

Ao contrário da análise anterior, no qual o tempo de inserção aumentava quanto maior o número de máquinas e fator de replicação, a análise feita na Tabela 4.8 mostra que há

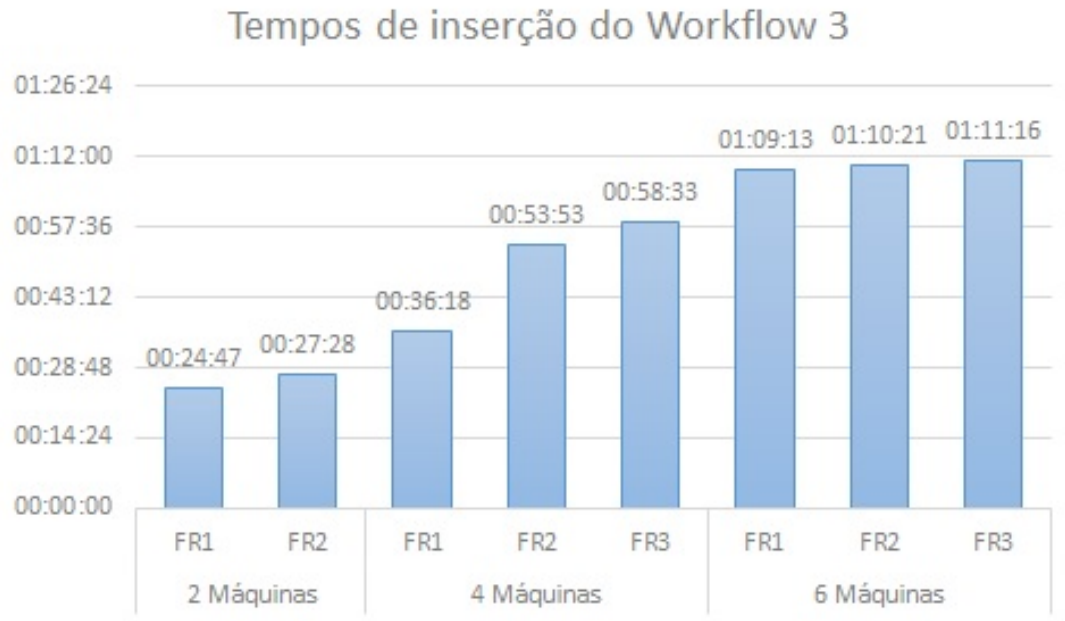

Figura 4.9: Tempos de Inserção do *Workflow* 3.

uma redução no tempo de extração conforme maior o número de máquinas e maior fator de replicação. A Figura 4.10 ajuda a visualizar tal fato. Na figura, pode-se observar que no *cluster* de 4 máquinas, por exemplo, com fator de replicação 1, o tempo de extração é de 20 minutos e 56 segundos, enquanto no fator de replicação 2 é de 17 minutos e com fator de replicação 3 o tempo é de 12 minutos e 8 segundos. Deve-se atribuir esse ganho de performance aos mecanismos de busca do Cassandra, uma vez que a busca é feita de forma distribuída, e quanto maior o número de réplicas, maior o número de possíveis origens para se buscar o dado, e mais rápido para se ter uma resposta do banco.

Analisando os dois extremos dos ambientes testados (*cluster* com 2 máquinas e fator de replicação 1 e *cluster* com 6 máquinas e fator de replicação 3), em suas execuções quentes, pode-se perceber que os tempos utilizados pela inserção e extração em comparação a execução dos *workflows* de Bioinformática testados, representam, no pior dos casos, menos de 10% do tempo total. Os gráficos das Figuras 4.12 e 4.11 demonstram o percentual dos tempos de execução contra os tempos das operações no SGBD quando ambos os tempos são somados.

O gráfico da Figura 4.11 é referente ao *cluster* de 6 máquinas com fator de replicação 3 e mostra que os tempos de inserção e extração do *workflow* 1 ocupa aproximadamente 10% do tempo total. Já os tempos de inserção e extração *workflow* 2 ocupa menos de 5% do tempo total e o *workflow* 3 ocupa cerca de 5%.

O gráfico da Figura 4.12 é referente ao *cluster* de 2 máquinas com fator de replicação 1 e

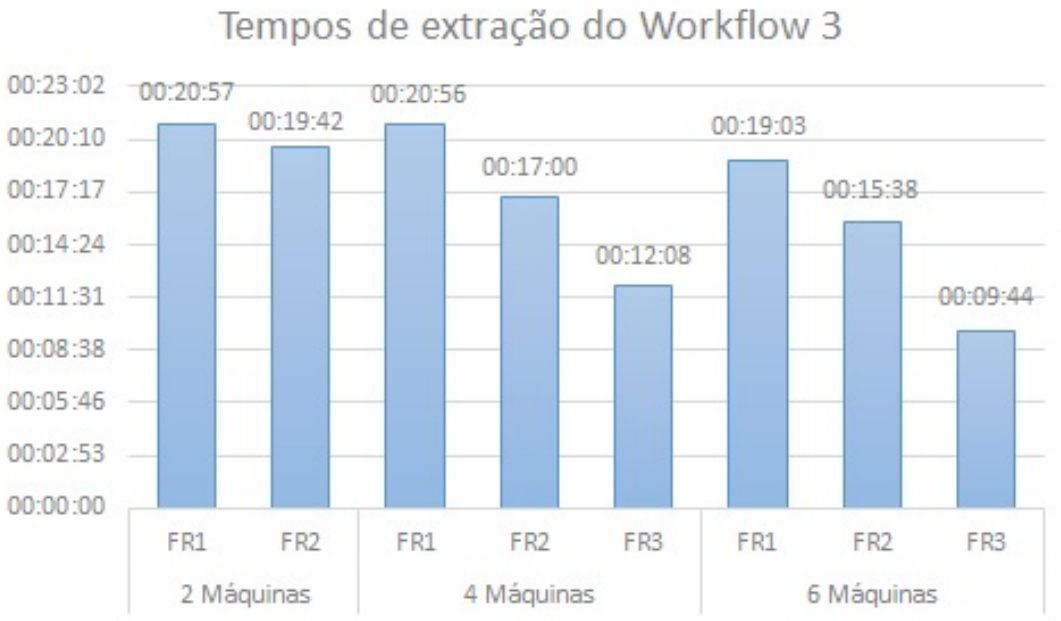

Figura 4.10: Tempos de Extração do *Workflow* 3.

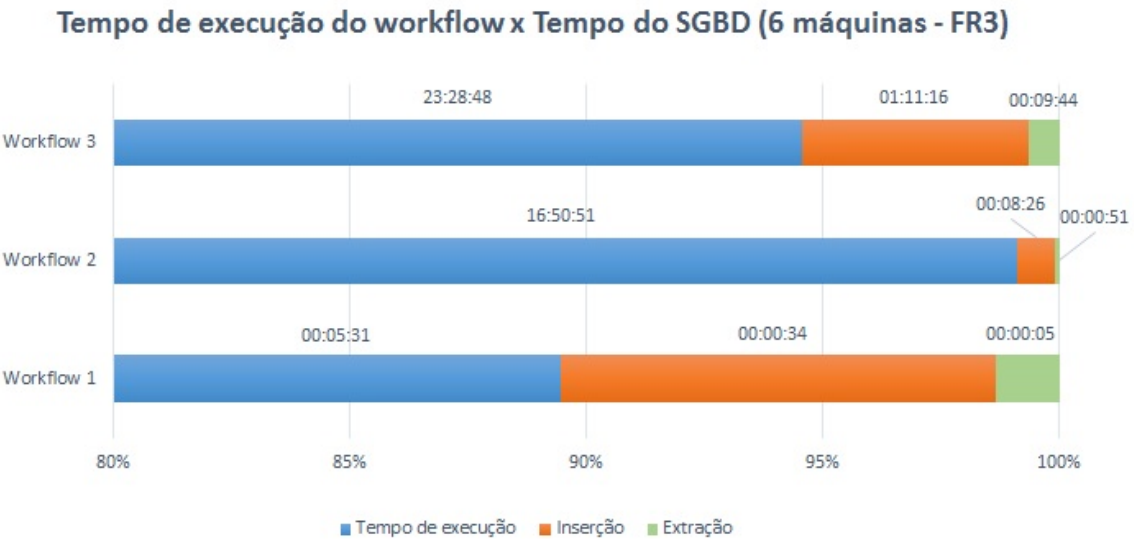

Figura 4.11: *Cluster* de 6 Máquinas e Fator de Replicação 3.

mostra que os tempos de inserção e extração do *workflow* 1 representam aproximadamente 5% do tempo total, enquanto os tempos de inserção e extração do *worfklow* 2 ocupam menos de 1% do tempo total, já os tempos *workflow* 3 representa menos de 5% do tempo total. Com isso é possível avaliar que o impacto em termo de tempo de processamento da utilização do SGBD Cassandra no ambiente de *workflow* de Bioinformática é pequeno.

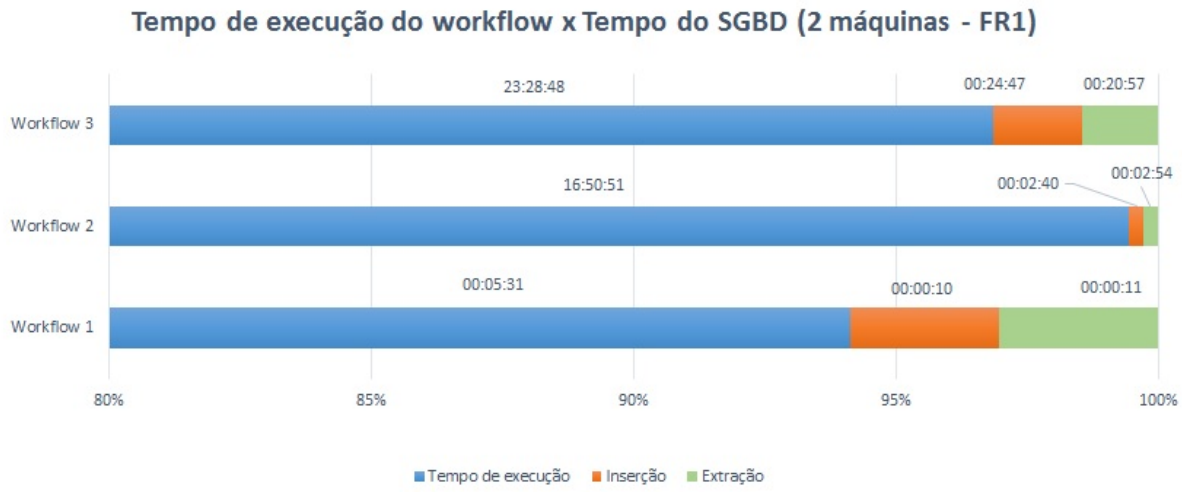

Figura 4.12: *Cluster* de 2 Máquinas e Fator de Replicação 1.

Capítulo 5

Conclusão e Trabalhos Futuros

O armazenamento de dados biológicos originados de *workflows* de Bioinformática é um desafio. Esta monografia apresentou a análise acerca da persistência de dados biológicos fazendo uso do sistema gerenciador de banco de dados NoSQL Cassandra. Com os resultados obtidos e análise apresentadas foi possível demonstrar diversas situações de uso do SGBD Cassandra.

De acordo com as análises apresentadas, em um ambiente com um maior número de máquinas com um fator de replicação maior, o tempo gasto para as operações de inserção é maior, ou seja, há um ganho de disponibilidade dos dados porém uma perda na performance para inserção. Nesses mesmos ambientes, no qual o número de máquinas e o fator de replicação é maior, foi mostrado também que o sistema gerenciador de banco de dados NoSQL Cassandra possui um menor tempo para leituras de dados. Com essas características é possível afirmar que o SGBD Cassandra é uma boa escolha para sistemas na qual existam poucas inserções e muitas leituras.

Em um ambiente de Bioinformática, no qual os arquivos seriam inseridos apenas uma vez e existem diversas requisições (leituras ao banco) de tais arquivos, pode-se dizer que o Cassandra é uma boa opção para armazenamento desses dados. As relações percentuais dos tempos de inserção e de extração dos dados, quando comparadas aos tempos de execução dos *workflows*, mostraram-se bem baixas, demonstrando que o banco de dados NoSQL Cassandra apresenta-se viável ao ser considerado um candidato ao banco de dados NoSQL no módulo para arquiteturas de gerenciadores de *workflow* em Bioinformática.

Um trabalho futuro a ser realizado é a reexecução dos *workflows* de forma distribuída, assim como também trabalhar com uma inserção no banco de dados de forma distribuída, a fim de diminuir o tempo de execução dos *workflows* e o tempo de inserção.

Acredita-se que a proposta aqui apresentada pode trazer um direcionamento no estudo para discutir um possível sistema gerenciador de banco de dados para armazenamento de dados da Bioinformática e em sistemas para proveniência de dados. Outros trabalhos

futuros podem consistir em estudar outros bancos de dados NoSQL, desenvolver aplicações semelhantes às realizadas nessa monografia e comparar a performance entre os dois sistema gerenciadores de banco de dados.

Referências

[1] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010. 19, 23

[2] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010. 17

[3] B. J. Haas, A. Papanicolaou, M. Yassour, M. Grabherr, P. D. Blood, J.Bowden, M. B. Couger, D. Eccles, B. Li, M. Lieber. De novo transcript sequence reconstruction from rna-seq using the trinity platform for reference generation and analysis. *Nature protocols*, 8(8):1494–1512, 2013. 34

[4] B. Langmead, S. L. Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 2012. 7, 34

[5] Joseph F Boland, Charles C Chung, David Roberson, Jason Mitchell, Xijun Zhang, Kate M Im, Ji He, Stephen J Chanock, Meredith Yeager, and Michael Dean. The new sequencer on the block: comparison of life technology's proton sequencer to an illumina hiseq for whole-exome sequencing. *Human genetics*, 132(10):1153–1163, 2013. 5

[6] C. A. Heuser. *Projeto de Banco de Dados*, volume 4. Bookman, 1998. 11

[7] C. Baudet. Uma abordagem para trimagem, verificação de contaminação e clusterização de sequências est. Master's thesis, Universidade Estadual de Campinas (Unicamp), 2006. 6

[8] C. Camacho, T. Madden, N. Ma, T. Tao, R. Agarwala, A. Morgulis. Blast command line applications user manual. 2008. 8, 35

[9] C. Trapnell L. Pachter, S. L. Salzberg. Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, 25(9):1105–1111, 2009. 7, 35

[10] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen and D. Wilkins. A comparison of a graph database and a relational database: A data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA, 2010. ACM. 14, 16

[11] Crucible II Group. *SEEDING SOLUTIONS Volume 1: Policy Options for Genetic Resources (People, Plants, and Patents Revisited)*, volume 1. International Development Research Center, 2000. 4

[12] D. Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, May 2008. 13

[13] D. W. Mount. *Bioinformatics: sequence and genome analysis*, volume 2. Cold spring harbor laboratory press New York:, 2001. 4

[14] DataStax. Cassandra documentation. `http://docs.datastax.com/en/cassandra/3.0`. Acessado em: 12-05-2016. xi, 23, 24, 25, 26, 27, 28, 29, 30

[15] Datastax. Cassandra introduction and key features. `http://pt.slideshare.net/planetcassandra/cassandra-introduction-features-30103666`. Acessado em: 28-05-2016. xi, 19, 20

[16] Datastax. Java driver for apache cassandra. `https://github.com/datastax/java-driver`. Acessado em: 10-09-2015. 36

[17] E. A. Brewer. Towards robust distributed systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 7–, New York, NY, USA, 2000. ACM. 13

[18] E. Brewer. Cap twelve years later: How the "rules"have changed. *Computer*, 45(2):23–29, Feb 2012. 14

[19] E. Hewitt. *Cassandra: the definitive guide*. "O'Reilly Media, Inc.", 2010. 16, 20, 21, 22, 30

[20] E. van Dijk, H. Auger, Y. Jaszczyszyn and C. Thermes. Ten years of next-generation sequencing technology. *Trends in genetics*, 30(9):418–426, 2014. 5

[21] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, Andrew and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008. 2, 16, 23

[22] F. Prosdocimi, G. Coutinho, E. Ninnecw, A. F. Silva, A. N. dos Reis, A. C. Martins, A. C. F. dos Santos, A. N. Júnior, F. Camargo Filho. Bioinformática: manual do usuário. *Biotecnologia Ciência e Desenvolvimento*, 29:12–25, 2002. 1

[23] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007. 2, 15

[24] H. Khazaei, M. Fokaefs, S. Zareian, N. Beigi-Mohammadi, B. Ramprasad, M. Shtern, P. Gaikwad, M. Litoiu. How do i choose the right nosql solution? a comprehensive theoretical and experimental survey. *Submitted to Journal of Big Data and Information Analytics (BDIA)*, 2015. xi, 17

[25] H. Lab. Fastx-toolkit. `http://hannonlab.cshl.edu/fastx_toolkit/`. Acessado em: 13-10-2015. 6

[26] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009. 9

[27] H. Saldanha, A. Araújo, C. Borges, E. Ribeiro, J. C. Setubal, M. E. Walter, M. Holanda, R. Gallon, R. Togawa. *Towards a hybrid federated cloud platform to efficiently execute bioinformatics workflows*. INTECH Open Access Publisher, 2012. 34

[28] H. V. F. Melo. Desenvolvimento de um pipeline para análise genômica e transcriptômica com base em web services. *Master's thesis, Universidade Federal de São Carlos*, 13:33, 2010. 5

[29] J. Aasman. Allegro graph: Rdf triple database. *Cidade: Oakland Franz Incorporated*, 2006. 17

[30] J. C. Anderson, J. Lehnardt, N. Slater. *CouchDB: the definitive guide*. "O'Reilly Media, Inc.", 2010. 16

[31] J. Gray, A. Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992. 12

[32] J. Han, E. Haihong, G. Le, J. Du. Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011. 20

[33] J. Orvis, J. Crabtree, K. Galens, A. Gussman, J. M. Inman, E. Lee, S. Nampally, D. Riley, J. P. Sundaram, V. Felix, and others. Ergatis: a web interface and scalable software system for bioinformatics workflows. *Bioinformatics*, 26(12):1488–1492, 2010. 18

[34] J. P. Latgé. Aspergillus fumigatus and aspergillosis. *Clinical microbiology reviews*, 12(2):310–350, 1999. 35

[35] J. Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pages 217–218. ACM, 2012. 17

[36] K. A. Wetterstrand. Dna sequencing costs: Data from the nhgri genome sequencing program (gsp). `www.genome.gov/sequencingcostsdata`. Acessado em: 17-06-2016. xi, 5

[37] K. Chodorow. *MongoDB: the definitive guide*. "O'Reilly Media, Inc.", 2013. 16

[38] L. George. *HBase: the definitive guide*. "O'Reilly Media, Inc.", 2011. 16

[39] M. G. Grabherr, B. J. Haas, M. Yassour, J. Z. Levin, D. A. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng. Trinity: reconstructing a full-length transcriptome without a genome from rna-seq data. *Nature biotechnology*, 29(7):644, 2011. 35

[40] M. Mattoso, J. Dias, F. Costa, D. de Oliveira, E. Ogasawara. Experiences in using provenance to optimize the parallel execution of scientific workflows steered by users. In *Workshop of Provenance Analytics*, 2014. 1

[41] M. Sawicki, G. Samara, M. Hurwitz, E. Passaro,. Human genome project. *The American journal of surgery*, 165(2):258–264, 1993. 4

[42] N. Rishe. *Database Design: The Semantic Modeling Approach*, volume 1. McGraw-Hill, 1992. 10

[43] O. Erling, I. Mikhailov. Virtuoso: Rdf support in a native rdbms. In *Semantic Web Information Management*, pages 501–519. Springer, 2010. 17

[44] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer and P. M. Rice. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic acids research*, 38(6):1767–1771, 2010. 8

[45] R. Aniceto, R. Xavier, V. Guimarães, et al. Evaluating the cassandra nosql database approach for genomic data persistency. *International Journal of Genomics*, 2015, 2015. 18

[46] R. C. Huacarpuma, M. Holanda, M. E. Walter. A conceptual model for transcriptome high-throughput sequencing pipeline. In *Brazilian Symposium on Bioinformatics*, pages 71–74. Springer, 2011. 17

[47] R. C. Team et. al. R: A language and environment for statistical computing. 2013. 8

[48] R. De Paula, M. T. Holanda, M. E. Walter, S. Lifschitz. Managing data provenance in genome project workflows. *BMC Bioinformatics*, 14(11):1–14, 2013. 5

[49] R. Hecht, S. Jablonski. Nosql evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing (CSC)*, pages 336–341. IEEE, Dec 2011. 14, 15, 16

[50] R. Ramakrishnan, J. Gehrke. *Sistemas de Gerenciamento de Banco de Dados*, volume 3. McGraw-Hill Interamericana, 2008. 10

[51] R. Schmieder, R. Schmieder. Quality control and preprocessing of metagenomic datasets. *Bioinformatics*, 27(6):863–864, 2011. 35

[52] R. W. Brito. Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa. Technical Report. Universidade de Fortaleza, 2010. 17

[53] S. A. Simon, J. Zhai, R. S. Nandety, K. P. McCormick, J. Zeng, D. Mejia, and B. C. Meyers. Short-read sequencing technologies for transcriptional analyses. *Annual review of plant biology*, 60:305–333, 2009. 5, 9

[54] T. Bloom, T. Sharpe. Managing data from high-throughput genomic processing: a case study. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1198–1201. VLDB Endowment, 2004. 17

[55] T. Carver, M. Berriman, A. Tivey, C. Patel, U. Böhme, B. G. Barrell, J. Parkhill, M. A. Rajandream. Artemis and act: viewing, annotating and comparing sequences stored in a relational database. *Bioinformatics*, 24(23):2672–2676, 2008. 18

[56] U. Röhm, J. Blakeley. Data management for high-throughput genomics. *arXiv preprint arXiv:0909.1764*, 2009. 17

[57] V. Guimaraes, F. Hondo, R. Almeida, H. Vera, M. Holanda, A. Araujo, M. E. Walter, S. Lifschitz,. A study of genomic data provenance in nosql document-oriented database systems. In *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, pages 1525–1531. IEEE, 2015. 14, 18

[58] V. Holt, M. Ramage, K. Kear, N. Heap. The usage of best practices and procedures in the database community. *Information Systems*, 49:163–181, 2015. 1

[59] Z. Ye, S. Li. A request skew aware heterogeneous distributed storage system based on cassandra. In *Computer and Management (CAMAN), 2011 International Conference on*, pages 1–5. IEEE, 2011. 18